Università degli Studi di Bologna

FACOLTÀ DI INGEGNERIA

Corso di Laurea Specialistica in
Ingegneria Informatica
Insegnamento: Reti di Calcolatori

AGGREGAZIONE DELLE QUERY PER SISTEMI PUB/SUB: IL PROGETTO SALES

Tesi di Laurea di:

Davide Candeloro

Relatore :

Chiar. mo Prof. Ing. **Antonio Corradi**

Correlatori:

Dott. Ing. **Mario Fanelli**

Dott. Ing. **Luca Foschini**

Sessione III

Anno Accademico 2011-2012

Parole chiave

SISTEMI CONTEXT-AWARE

MODELLO PUBLISH/SUBSCRIBE

QUERY MERGING ADATTATIVO

RIDUZIONE CARICO CPU

RIDUZIONE BANDA

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

I sistemi mobili rappresentano una classe di sistemi distribuiti caratterizzata dalla presenza di dispositivi portatili eterogenei quali PDA, laptop e telefoni cellulari che interagiscono tra loro mediante una rete di interconnessione wireless. Una classe di sistemi mobili di particolare interesse è costituita dai sistemi basati sul modello di interazione publish/subscribe. Secondo tale schema, i nodi all'interno di una rete possono assumere due ruoli differenti: i produttori di informazione, chiamati publisher, ed i consumatori di informazione, chiamati subscriber. Tipicamente, l'interazione tra essi è mediata da un gestore di eventi che indirizza correttamente le informazioni ricevute dai publisher verso i subscriber interessati, sulla base degli interessi espressi da questi ultimi tramite sottoscrizioni. Nella progettazione di sistemi mobili, a differenza di quelli tradizionali basati su nodi fissi, bisogna tenere conto di problemi quali la scarsa capacità computazionale dei dispositivi e la limitata larghezza di banda delle reti wireless. All'interno di tale ambito, stanno recentemente assumendo sempre maggiore importanza i sistemi context-aware, ovvero sistemi mobili che sfruttano i dati provenienti dall'ambiente circostante e dai dispositivi stessi per adattare il proprio comportamento e notificare agli utenti la presenza di informazioni potenzialmente utili. Nello studio di questi sistemi, si è notato che i nodi che si trovano nella stessa area geografica generano tipicamente delle sottoscrizioni che presentano tra loro un certo grado di similarità e coperture parziali o totali.

Il gruppo di ricerca del DEIS dell'Università di Bologna ha sviluppato un'infrastruttura di supporto per sistemi mobili context-aware, chiamata

SALES. Attualmente il sistema progettato non considera le similarità delle sottoscrizioni e quindi non sfrutta opportunamente tale informazione. In questo contesto si rende necessario l'adozione di opportune tecniche di aggregazione delle sottoscrizioni atte ad alleggerire la computazione dei nodi mobili e le comunicazioni tra loro. Il lavoro presentato in questa tesi sarà finalizzato alla ricerca, all'adattamento ed all'implementazione di una tecnica di aggregazione delle sottoscrizioni. Tale tecnica avrà lo scopo di rilevare e sfruttare le similarità delle sottoscrizioni ricevute dal sistema al fine di ridurne il numero; in questo modo, quando un nodo riceverà un dato, il processo di confronto tra l'insieme delle sottoscrizioni memorizzate e il dato ricevuto sarà più leggero, consentendo un risparmio di risorse computazionali. Inoltre, adattando tali tecniche, sarà possibile modulare anche il traffico dati scaturito dalle risposte alle sottoscrizioni.

La struttura di questa tesi prevede un primo capitolo sui sistemi context-aware, descrivendone le principali caratteristiche e mettendo in luce le problematiche ad essi associate.
Il secondo capitolo illustra il modello di comunicazione Publish/Subscribe, modello di riferimento per i moderni sistemi context-aware e per i sistemi mobili in generale.
Il terzo capitolo descrive l'infrastruttura SALES sulla quale si è progettata, implementata e testata la soluzione proposta in questa tesi.
Il quarto capitolo presenta le principali tecniche di aggregazione delle sottoscrizioni e spiega come possono essere adattate al contesto di questa tesi.
Il quinto capitolo effettua l'analisi dei requisiti per comprendere meglio il comportamento della soluzione; seguono la progettazione e l'implementazione della soluzione su SALES.
Infine, il sesto capitolo riporta in dettaglio i risultati ottenuti da alcuni degli esperimenti effettuati e vengono messi a confronto con quelli rilevati dal sistema di partenza.

Capitolo 1

Sistemi context-aware

Negli ultimi anni le tecnologie di comunicazione hanno subito un notevole sviluppo, consentendo una sempre maggiore disponibilità di connettività nei luoghi che più spesso visitiamo. Mentre un tempo l'accesso alla rete era possibile esclusivamente tramite dispositivi fissi, tipicamente collocati all'interno delle abitazioni, ai giorni d'oggi è possibile usufruire di innumerevoli servizi in completa mobilità, senza preoccuparsi di dove ci si trova. Questa notevole innovazione tecnologica ha portato ad una diffusione sempre maggiore di dispositivi mobili, in grado di fornire accesso a servizi prima disponibili solo tramite rete fissa e dotati di una sempre più elevata capacità di elaborazione. Al sorgere di questo scenario è stato necessario rivedere in una nuova ottica i precedenti paradigmi di comunicazione.

I sistemi distribuiti tradizionali si appoggiano ad un ambiente prevalentemente statico in cui le informazioni provenienti dall'ambiente circostante non ricoprono un'importanza fondamentale. Inoltre tali sistemi erano collegati fra loro da una rete fissa in grado di offrire prestazioni elevate e stabili nel tempo.

Tutte queste considerazioni non sono più valide quando si fa riferimento a sistemi mobili e si è quindi ritenuto opportuno introdurre nuovi paradigmi di comunicazione. Essi dovranno permettere a tali sistemi di operare in maniera dipendente dall'ambiente in cui si trovano, collezionando e sfruttando le informazioni che l'ambiente mette a disposizione e fornen-

do un elevato grado di robustezza ed affidabilità di fronte ad eventuali disconnessioni dei dispositivi dalla rete.

I sistemi che sfruttano questo nuovo paradigma sono chiamati *context-aware* e su di essi si baserà questo capitolo. Il Paragrafo 1.1 introduce brevemente i sistemi distribuiti. Nel Paragrafo 1.2 si è ritenuto fondamentale spiegare cosa sono i sistemi mobili, soffermandosi sulle diverse modalità di comunicazione e sulle problematiche da affrontare. Il Paragrafo 1.3 definisce il significato della parola 'contesto' in ambito informatico e analizza i suoi aspetti principali. Il Paragrafo 1.4 descrive i principali tipi e le caratteristiche delle applicazioni context-aware. Il Paragrafo 1.5 definisce la distribuzione dei dati di contesto e fornisce i principali requisiti che essa deve soddisfare. Il Paragrafo 1.6 propone un modello architetturale per la distribuzione dei dati di contesto. Infine, il Paragrafo 1.7 motiva i lavoro svolto esponendo gli obiettivi della tesi.

1.1 Sistemi distribuiti

Un sistema si dice distribuito se costituito da diversi componenti collocati in una rete che comunicano e coordinano le loro azioni solo attraverso lo scambio di messaggi [**Cou12**]. Questa semplice definizione copre l'intero range di sistemi nei quali è possibile sfruttare l'utilizzo di computer connessi ad una rete. Tale definizione di sistema distribuito porta alle seguenti conseguenze:

- *Concorrenza*: in una rete di computer, normalmente si eseguono programmi concorrentemente. Ogni computer può svolgere il proprio lavoro, condividendo risorse quali pagine web o file quando necessario.

- Nessun *clock globale*: quando i programmi necessitano di collaborare, essi coordinano le loro azioni mediante lo scambio di messaggi. La coordinazione spesso si basa sull'idea condivisa del tempo in cui avvengono le azioni dei programmi. Ci sono però dei forti limiti all'accuratezza con la quale i computer connessi ad una rete possono

sincronizzare i loro clock, perciò non può esistere un'unica nozione di tempo esatto. Questa è una diretta conseguenza del fatto che l'unica maniera per comunicare in una rete è attraverso lo scambio di messaggi.

- *Failure indipendenti*: tutti i sistemi possono fallire ed è responsabilità del progettista di sistema pensare alle conseguenze di possibili fallimenti. I fallimenti nella rete portano all'isolamento dei computer, ma questo non significa che i computer interrompono la loro esecuzione. Infatti i programmi non sono in grado di stabilire se la rete è fallita oppure è insolitamente lenta. In maniera simile, i fallimenti dei computer, o l'inaspettata terminazione dei programmi in qualche parte del sistema (crash), non sono immediatamente resi noti agli altri componenti con i quali esso comunica. Ogni componente del sistema può fallire indipendentemente dagli altri.

Il primo motivo che spinge alla costruzione ed all'utilizzo di sistemi distribuiti deriva dal desiderio di condividere risorse. Il termine 'risorsa' è abbastranza astratto, ma caratterizza al meglio tutte le cose che possono essere vantaggiosamente condivise in una rete di computer. Esso comprende sia i componenti hardware, quali dischi e stampanti, che entità software, quali file, database e qualsiasi tipo di oggetto che racchiuda informazioni.

1.2 Sistemi mobili

La nascita dei sistemi mobili si può ricondurre alla progressiva diffusione dei personal computer portatili, a sua volta legata ai progressi tecnologici che hanno portato ad una diminuzione continua delle dimensioni dei componenti hardware. Parallelamente è nato il paradigma del *mobile computing*, che designa in modo generico le tecnologie di elaborazione e di accesso ai dati prive di vincoli sulla posizione dell'utente e sulle apparecchiature coinvolte. Questo paradigma considera tre differenti aspetti: la comunicazione mobile, l'hardware mobile ed il software mobile. In questa

sezione ci soffermeremo solo sull'aspetto della comunicazione all'interno di sistemi mobili, utile alla comprensione del lavoro di tesi.

Le tecnologie di comunicazione in ambienti mobili vengono chiamate tecnologie wireless e si distinguono dalle tecnologie utilizzate nei sistemi distribuiti tradizionali per il fatto di non richiedere interconnessioni fisiche fra le entità coinvolte nella comunicazione. La comunicazione tipica fra due computer desktop avviene tramite l'invio di segnali elettrici propagati attraverso cavi fisici, i quali rappresentano dunque il mezzo d'interconnessione (reti *wired*). In uno scenario mobile, il mezzo è invece l'aria ed i dispositivi mobili comunicano fra di loro generalmente tramite l'emissione di onde radio.

La rete più comunemente utilizzata in questo ambito è la *wireless local area network* (WLAN), una rete locale in grado di collegare due o più dispositivi utilizzando una specifica tecnologia senza fili e che solitamente è in grado di garantire l'accesso alla rete globale Internet attraverso l'utilizzo di un particolare dispositivo chiamato *Access Point* (AP). La tecnologia wireless attualmente utilizzata in queste reti è descritta dallo standard IEEE 802.11 e prende il nome di Wi-Fi.

I nodi che compongono la rete wireless sono in grado di comunicare fra loro sfruttando solitamente due diverse modalità: infrastrutturata e ad-hoc. La prima prevede che ogni nodo sia connesso ad un AP, il quale costituisce un punto di centralizzazione nella comunicazione tra nodi e fornisce, eventualmente, l'accesso verso la rete esterna. La seconda permette invece una comunicazione diretta tra due nodi, senza bisogno di alcuna infrastruttura di supporto alla comunicazione. Di seguito saranno analizzate queste due modalità di comunicazione, sottolineando i maggiori pregi e diffetti di entrambe, nonché il loro possibile campo di utilizzo.

1.2.1 Modalità infrastruttura

La gestione della rete wireless attraverso questa modalità prevede l'installazione di un AP a cui i diversi nodi mobili potranno collegarsi per comunicare fra di loro o verso l'esterno della rete. Viene così a crearsi una topologia di rete a stella dove l'AP rappresenta il nodo centrale, in cui dovranno transitare tutti i dati inviati dai dispositivi mobili, come mostra la Figura 1.1. I nodi collegati all'AP potranno muoversi all'interno del suo

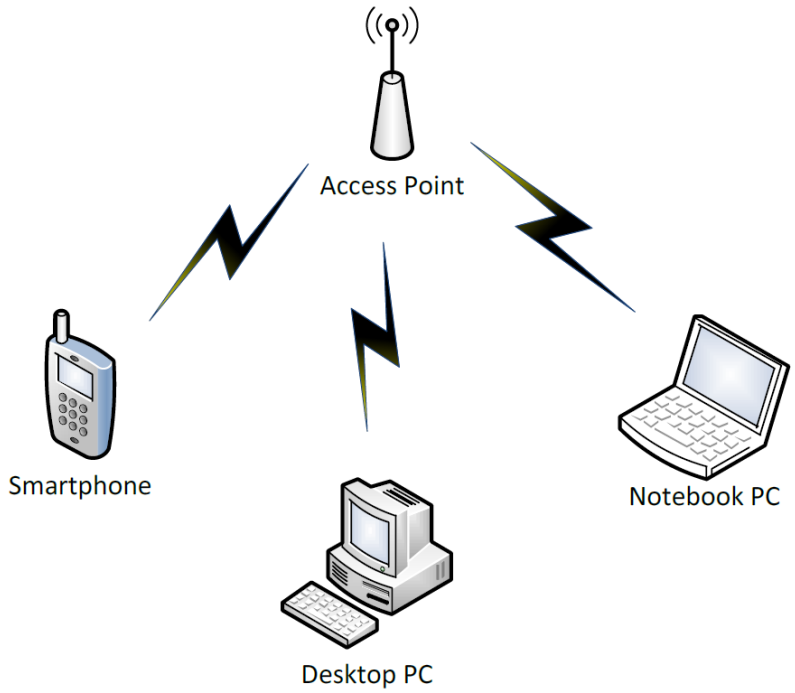

Figura 1.1: Rete wireless infrastrutturata.

campo di trasmissione senza correre il rischio di perdere il collegamento con lo stesso: in tal modo può essere garantita la mobilità dei dispositivi durante la fruizione dei servizi. È importante sottolineare che questi dispositivi sono spesso eterogenei e ciò non deve influire in alcun modo sulla loro comunicazione.

I nodi possono comunicare fra loro esclusivamente attraverso l'utilizzo dell'AP, che fa da intermediario nella comunicazione fra i pari ed anche da

ponte verso altre reti wireless o fisse. È possibile far comunicare diversi AP tra di loro attraverso l'ausilio di una rete fissa di comunicazione, creando così un'unica rete wireless logica più ampia. E' possibile collegare l'AP ad una rete fissa per consentire anche il colloquio con macchine residenti su reti wired (solitamente desktop PC).

Un approccio di questo tipo risulta più semplice da realizzare, poichè la maggior parte delle funzionalità di comunicazione sono implementate all'interno dell'AP. La modalità infrastruttura possiede inoltre considerevoli vantaggi in termini di scalabilità e di gestione centralizzata della sicurezza. Queste reti sono tipicamente utilizzate in ambiente domestico o lavorativo proprio per la loro semplicità di configurazione e utilizzo.

Il principale svantaggio è dovuto invece alla necessità di installare un AP per poter comunicare, nonchè al costo dello stesso che, nel caso di ampie installazioni, potrebbe essere considerevole. La presenza obbligatoria dell'AP comporta una minore flessibilità di questa soluzione rispetto alla modalità ad-hoc, la quale è in grado di formare una rete senza bisogno di alcuna infrastruttura di supporto.

1.2.2 Modalità ad-hoc

La modalità wireless ad-hoc rappresenta una tecnologia che permette ai dispositivi mobili di comunicare direttamente fra di loro, senza dover contare sulla presenza di un intermediario o di un'infrastruttura di rete preesistente. Un dispositivo che intende comunicare utilizzando questa modalità dovrà effettuare un'operazione di *discover* per individuare i pari disponibili. Il pari dovrà preventivamente rendersi visibile ai dispositivi circostanti (*discoverable*) per poter essere contattato.

Un esempio di paradigma di rete basato su comunicazioni ad-hoc è quello del Mobile Ad-hoc NETwork (MANET). La topologia di questi sistemi mobili non è fissa ma creata dinamicamente dai partecipanti alla comunicazione. La Figura 1.2 mostra un esempio di tale modalità.

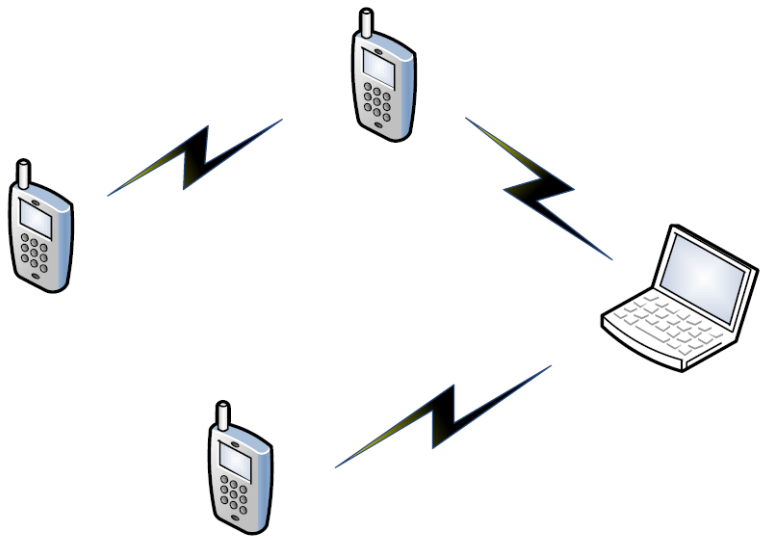

Figura 1.2: Rete wireless ad-hoc.

Una rete di questo tipo risulta più complessa di una rete wireless infrastrutturata, in quanto ogni nodo partecipante dovrà implementare dei meccanismi con cui poter effettuare l'accesso al canale di comunicazione. Tutto quello che nella modalità infrastruttura è controllato dall'AP deve essere gestito dai dispositivi mobili, introducendo una maggiore difficoltà implementativa. Un'altra importante considerazione è che ora l'intera infrastruttura è mobile e la gestione della stessa è in mano ai singoli dispositivi, comportando maggiori difficoltà nella rilevazione di errori e nell'amministrazione della rete. Una realizzazione di questo tipo dovrà anche fare i conti con l'instradamento dei messaggi; i singoli dispositivi dovranno comportarsi come dei router, instradando non solo i messaggi destinati a loro, ma anche agli altri nodi della rete. Tale complessità influisce anche sul consumo di risorse dei singoli dispositivi e sul conseguente consumo di energia, ricordando che molti di essi hanno tipicamente un numero limitato di risorse ed un'autonomia energetica bassa.

I vantaggi di questa soluzione sono rappresentati dalla possibilità di poter comunicare in ogni momento con altri dispositivi, anche eterogenei, indipendentemente dal luogo in cui ci si trova. Ciò costituisce il motto

coniato alla nascita di questa modalità di comunicazione: 'anytime and anywhere'. Nelle situazioni in cui si vuole garantire un'elevata disponibilità della comunicazione questa è sicuramente la modalità più opportuna da utilizzare.

1.2.3 Reti miste

Le reti miste, o ibride, si prefiggono lo scopo di combinare le due modalità viste in precedenza, andando a realizzare un'infrastruttura composta da due parti. La Figura 1.3 ne mostra un esempio. La prima parte è composta

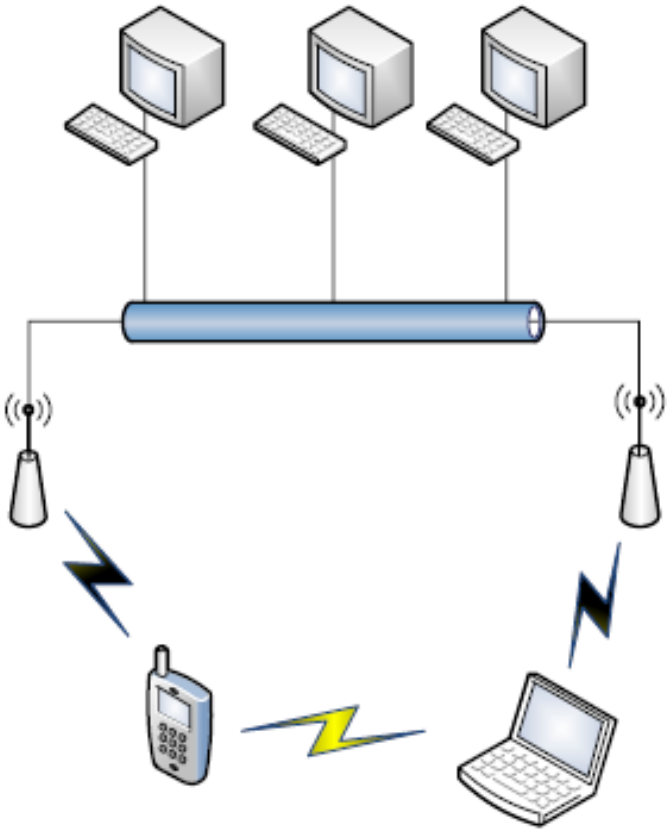

Figura 1.3: Rete mista.

da una rete ad-hoc, caratterizzata dalla presenza esclusiva di nodi mobili. La seconda parte è una rete composta da un numero variabile di AP, eventualmente collegati ad una rete fissa per consentire l'accesso alla rete

globale da parte dei dispositivi. In questo scenario gli AP sono degli intermediari per la comunicazione con la rete fissa e non sono tenuti a svolgere operazioni che favoriscano la comunicazione fra i nodi mobili della rete, i quali già posseggono tutte le funzionalità per comunicare fra di loro.

Con reti di questo tipo è possibile sfruttare i vantaggi di entrambe le soluzioni, dipendentemente dallo scenario di utilizzo. Si può, ad esempio, ottenere scalabilità differenziando il traffico fra interno e esterno alla rete. E' anche possibile garantire una maggiore copertura sfruttando la modalità ad-hoc per estendere i servizi messi a disposizione dalla rete wireless fissa. Si possono sfruttare anche diverse tecnologie di comunicazione per le modalità infrastruttura e ad-hoc, facendo sì che un maggior numero di dispositivi sia abilitato alla comunicazione. Ad esempio, un dispositivo potrebbe utilizzare la tecnologia Wi-Fi in modalità infrastruttura e quella Bluetooth in modalità ad-hoc, rendendo così opzionale la possibilità di una comunicazione ad-hoc tramite Wi-Fi, la quale non è disponibile in tutti i dispositivi mobili. Questa tesi si colloca in un ambiente di questo tipo.

1.2.4 Problematiche nei sistemi mobili

I sistemi di questo tipo, rispetto all'approccio tradizionale, presentano diverse problematiche e limitazioni, che con il tempo si sta cercando di attenuare per consentire una maggiore efficienza ed affidabilità. Verranno di seguito elencate le problematiche più importanti, ordinate a partire dalle situazioni di più basso livello, prendendo come riferimento lo stack di protocolli ISO/OSI:

- *Interferenze di trasmissione*: le informazioni non viaggiano più attraverso mezzi fisici protetti come i cavi, perciò possono essere maggiormente sensibili agli eventi atmosferici e alla conformazione del territorio, rispetto a soluzioni fisse.

- *Larghezza di banda*: rispetto alla soluzione fissa, la quale presenta solitamente una larghezza di banda stabile ed ampia, i sistemi mobili sono spesso caratterizzati da una larghezza di banda ridotta e molto variabile.

- *Consumo di energia*: i dispositivi appartenenti a reti mobili sono per lo più dispositivi con un'autonomia limitata, contrariamente a quanto avviene con i computer desktop. L'accesso alla rete comporta un consumo abbastanza elevato di energia, che va a compromettere la durata della batteria. Questo aspetto è fondamentale e deve essere tenuto in maggiore considerazione dai progettisti dei dispositivi, in quanto ad oggi molti di essi presentano un'autonomia energetica non sufficiente per un utilizzo frequente.

- *Sicurezza*: nei sistemi mobili, quando si accede ad una rete pubblica, si è esposti ad eventuali attacchi provenienti da altri dispositivi, cosa che avviene con minore probabilità nelle reti fisse.

Vi sono poi altre due proprietà che una rete wireless deve tenere in considerazione, così come avviene anche per le reti fisse:

- *Scalabilità*: il sistema deve dipendere il meno possibile dalla quantità di nodi presenti nella rete, per poter realizzare vaste installazioni senza che nascano problematiche di gestione e con lo scopo di ridurre l'overhead di comunicazione. La scalabilità è garantita soprattutto nelle soluzioni infrastrutturate, mentre nella modalità ad-hoc è più problematica all'aumentare del numero di nodi, a causa della complessità riversata sui dispositivi mobili e alla maggiore dinamicità del sistema.

- *Interoperabilità*: i dispositivi fra di loro eterogenei devono poter comunicare senza alcun problema. Questa caratteristica può essere presente sia nella modalità infrastruttura che in quella ad-hoc, ma la sua realizzazione dipende fortemente dal software utilizzato.

1.3 Definizione di contesto e di qualità di contesto

I sistemi context-aware derivano dai sistemi distribuiti ma, a differenza di essi, sfruttano il concetto di 'contesto' per adattare la loro esecuzione. Per

capire cosa sono i sistemi context-aware è prima di tutto necessario definire il significato della parola 'contesto' in ambito informatico e strutturarlo in base alle diverse tipologie che esso può assumere. Molte sono state le interpretazioni su questo argomento ma sicuramente quella di Chen e Kotz [**Che00**] si distingue in quanto è in grado di coprire i principali aspetti del contesto con una classificazione piuttosto diretta. Il contesto viene definito come l'insieme delle impostazioni e degli stati ambientali che determina il comportamento di un'applicazione o in cui avviene un evento applicativo di interesse per l'utente. Da questa definizione emergono due aspetti fondamentali del contesto:

- un primo aspetto include le caratteristiche dell'ambiente circostante, in base alle quali varia il comportamento dell'applicazione mobile;

- un secondo aspetto include la notifica degli eventi all'utente interessato e si presenta rilevante ma non necessario per l'applicazione.

In tal modo si può scomporre il contesto in base a questi due aspetti, individuando nel primo un contesto attivo, critico per il funzionamento ottimale dell'applicazione, e nel secondo un contesto passivo, rilevante ma non critico come il precedente poiché l'applicazione non è tenuta ad adattarsi a questo tipo di informazioni. Il fatto che uno specifico dato di contesto rientri in una o nell'altra categoria è a discrezione del tipo di applicazione che si intende realizzare, perciò uno stesso tipo di dato può venire inteso in modo diverso da due distinte applicazioni.

Inoltre Chen e Kotz [**Che00**] descrivono il contesto come uno spazio a quattro dimensioni composto da:

- *Contesto di computing*: riguarda tutti gli aspetti tecnici relativi alle risorse e alle capacità computazionali. Tale categoria ha un duplice scopo. Primo, esprimere tutte le eterogeneità presenti solitamente in ambienti mobili, come le diverse capacità e connettività dei dispositivi; secondo, considerare le diverse risorse che un dispositivo mobile incontra mentre si muove.

- *Contesto fisico*: raggruppa tutti gli aspetti che rappresentano il mondo reale e che sono accessibili tramite sensori posti vicino al nodo. Un esempio rilevante è costituito dalla posizione del dispositivo/utente; altri aspetti includono le condizioni del traffico, la velocità dell'utente, la temperatura e molti altri. Per sua natura il contesto fisico è intrinsecamente molto incline ad errori di misura a causa dell'imprecisione e della natura stocastica dei processi fisici.

- *Contesto temporale*: cattura la dimensione tempo di qualsiasi attività nel sistema come l'orario, il giorno della settimana, il mese e la stagione. Inoltre voci di contesto possono essere sporadiche o periodiche: gli eventi sporadici sono gli eventi che avvengono occasionalmente o una volta soltanto; gli eventi periodici invece si presentano in modo ripetibile e predicibile.

- *Contesto utente*: contiene tutti gli aspetti di alto livello legati alla dimensione sociale dell'utente come il profilo utente, le persone vicino e la situazione sociale corrente. Difatti i sistemi context-aware sono dei sistemi mobili distribuiti che prevedono la presenza di più utilizzatori finali. Perciò ogni nodo ha sia un contesto individuale derivante dalla vista egocentrica, sia un contesto sociale derivante dalla consapevolezza di essere un attore dell'intero sistema.

Considerando queste quattro dimensioni si possono realizzare diversi comportamenti context-aware allo scopo di adattare i servizi per soddisfare l'utente finale e per adeguarsi alle caratteristiche dell'ambiente di esecuzione corrente. Per raggiungere tale obiettivo, assume un aspetto fondamentale la qualità dei dati di contesto perché possono compromettere la correttezza delle operazioni di adattamento. Infatti, se da un lato è noto che i sensori fisici introducono errori e approssimazioni, dall'altro lato anche l'utilizzo di sensori virtuali (come i dati estratti da un database) non assicurano la correttezza totale, nonostante tali sensori forniscano dati più raffinati rispetto a quelli fisici. Di conseguenza la nozione di *Qualità del Contesto* (QoC), tipicamente definita come l'insieme dei parametri

che esprimono i requisiti di qualità e le proprietà dei dati di contesto, è di fondamentale importanza per controllare e gestire tutte le possibili imprecisioni del contesto stesso.

Ora che è stata data una definizione di contesto è importante comprendere come esso venga utilizzato in modo efficiente all'interno dei sistemi che sfruttano tali informazioni; di questo si occuperà il prossimo paragrafo.

1.4 Context-aware computing

Un sistema viene definito context-aware se 'utilizza il contesto per fornire informazioni e/o servizi rilevanti per l'utente, dove la rilevanza dipende dalle azioni che l'utente sta eseguendo in quel momento' [**Dey99**]. Riprendendo la differenza identificata in precedenza fra contesto attivo e passivo è possibile notare come tale definizione di sistema context-aware sia molto generale. Essa include il fatto che non sempre tali sistemi adattano il proprio comportamento in base al contesto, ma possono in realtà solamente notificare all'utente le informazioni, gestendo dunque i dati di contesto come entità passive. In tal modo si da maggiore importanza alla raccolta e al riconoscimento dei dati di contesto da parte di questi sistemi più che alle azioni che si faranno sugli stessi.

Per agevolare la comprensione è utile suddividere in categorie le applicazioni context-aware in base alla loro funzionalità. Schilit et al. hanno ideato una tassonomia di tali applicazioni basata su due livelli ortogonali [**Sch94**]: il primo livello considera l'obiettivo stesso dell'applicazione, che può essere quello di ottenere delle informazioni o di eseguire un comando; il secondo livello specifica invece la modalità con cui viene conseguito tale obiettivo, se manualmente o automaticamente. Come mostra la Tabella 1.1, questa suddivisione porta alla presenza di quattro differenti tipologie di applicazione:

		Scopo	
		Ottenere informazioni	Eseguire comandi
Modalità	Manuale	Proximate selection	Context commands
	Automatico	Automatic contextual reconfiguration	Context-triggered actions

Tabella 1.1: Tassonomia delle applicazioni context-aware secondo Schilit.

- *Proximate selection*: applicazioni che ritrovano le informazioni per l'utente, in base al contesto disponibile, presentandogli una lista di risorse o luoghi dove hanno la priorità elementi nelle più strette vicinanze. Sarà poi compito dell'utente scegliere la risorsa o il luogo che più preferisce. In tal modo si favorisce la scelta dell'utilizzatore prediligendo la comodità nel raggiungere l'entità d'interesse, qualora questo sia il suo scopo. Ad esempio un'applicazione di questo tipo consentirebbe la ricerca di una stampante nella zona; l'applicazione potebbe mostrare tutte le stampanti disponibili in ordine di distanza crescente dall'utente, favorendo così la selezione dell'entità più vicina;

- *Automatic contextual reconfiguration*: applicazioni che recuperano le informazioni per l'utente, basandosi sul contesto attuale per effettuare un binding automatico ad una risorsa disponibile. Un esempio potrebbe riguardare il collegamento automatico ad una stampante ritrovata nel contesto attuale per rendere più rapide operazioni di stampa emesse dal dispositivo. Ciò renderebbe non necessaria alcuna selezione delle stampanti disponibili da parte dell'utente;

- *Contextual commands*: applicazioni che eseguono comandi su richiesta dell'utente in modo manuale, in base al contesto disponibile. Tali servizi possono essere disponibili per l'utente in base al contesto in cui esso si trova e possono anche modificare il loro comportamento in base ad esso, fornendo così un maggiore grado di affidabilità di fronte a determinate situazioni. Utilizzando questo genere di appli-

cazioni, un utente che si trova in un cinema potrebbe essere in grado di visualizzare gli orari degli spettacoli disponibili ed eventualmente effettuare una prenotazione direttamente dal proprio dispositivo mobile;

- *Context-triggered actions*: applicazioni in grado di eseguire comandi in modo automatico basandosi su determinate situazioni di contesto. Ciò porta alla creazione di clausole if-then che permettono all'applicazione di adattarsi in base alle informazioni di contesto collezionate, senza che l'utente ne debba obbligatoriamente essere cosciente.

È inoltre possibile fare riferimento ad una rappresentazione meno dettagliata di tali categorie considerando esclusivamente le applicazioni che utilizzano in maniera attiva le informazioni di contesto adattandosi ad esse (context-awareness attiva) e quelle che invece utilizzano le stesse per effettuare delle semplici notifiche all'utente (context-awareness passiva). In questo ambito le applicazioni che in precedenza avevano lo scopo di ottenere informazioni possono essere considerate applicazioni a contesto passivo, mentre le altre sono considerate applicazioni a contesto attivo.

1.5 Distribuzione dei dati di contesto

Formalmente si definisce distribuzione dei dati di contesto come quella funzione dell'infrastruttura (distribuita) che rende possibile l'iniezione dei dati di contesto nel sistema e la consegna automatica di tali dati a tutte le entità che hanno espresso interesse per essi. Per supportare efficacemente i servizi context-aware in una rete wireless molto ampia, il gruppo di ricerca del DEIS[1] **[Bel12]** sostiene che la distribuzione dei dati di contesto debba soddisfare diversi requisiti principali:

- *Disaccoppiamento*: la distribuzione dei dati di contesto deve instradare i dati prodotti a tutti gli interessati. Per favorire la scalabilità del sistema e la disponibilità del contesto, la produzione e il consumo

[1]Dipartimento di Elettronica, Informatica e Sistemistica dell'Università di Bologna

di tali dati dev'essere possibile anche ad istanti temporali differenti (disaccoppiamento temporale), inoltre i produttori e i consumatori non devono conoscersi a vicenda (disaccoppiamento spaziale). In altre parole la comunicazione deve essere asincrona e anonima tra i produttori e i consumatori del contesto, come nei tradizionali sistemi pub/sub, i quali saranno discussi nel Capitolo 2.

- *Eterogeneità* e *mobilità*: la distribuzione dei dati di contesto deve supportare il funzionamento in scenari in cui sono presenti dispositivi eterogenei. I nodi mobili che richiedono i servizi context-aware si collegano e scollegano, talvolta in maniera random, introducendo variazioni improvvise sulle esigenze del contesto. Perciò la distribuzione dei dati di contesto deve prontamente adattarsi alla mobilità, così da distribuire solo i dati di contesto richiesti sul momento. Allo stesso tempo, questa funzione deve lavorare con sistemi eterogenei, incluso nodi con differenti capacità computazionali, standard wireless e modalità wireless. Perciò, risulta fondamentale adattarsi alle risorse disponibili sul momento per evitare la saturazione del sistema.

- *Principio di località*: la distribuzione dei dati di contesto deve introdurre, preservare e applicare *scope* differenziati di visibilità dei dati. Infatti, questi ultimi hanno spesso una limitata visibilità, dipendentemente dal principio di località. In altre parole, i dati di contesto hanno intrinsecamente uno scope di visibilità che la distribuzione deve applicare per evitare un inutile overhead di gestione.

- *Qualità del contesto (QoC)*: la distribuzione dei dati di contesto deve applicare i vincoli di QoC per una gestione corretta del sistema. Tradizionalmente la QoC viene definita come l'insieme dei parametri che esprimono proprietà e requisiti di qualità per i dati di contesto (ad esempio precisione, freschezza, affidabilità, etc.); tuttavia, si sta diffondendo sempre più l'idea di considerare la QoC non solo sui dati di contesto ma anche sulla distribuzione stessa di essi. Tale visione della QoC, implementata nell'infrastruttura SALES descritta

nel Capitolo 3, consente di considerare delle garanzie temporali e di affidabilità per la consegna dei dati. Infatti la distribuzione dei dati di contesto spesso sfrutta un'infrastruttura wireless best-effort, ovvero senza garanzie di consegna, che potrebbe introdurre ritardi e perdite, portando perciò ad un'inaccuratezza ulteriore del contesto finale ricevuto dai dispositivi mobili.

- *Data management*: la distribuzione dei dati di contesto deve gestire il ciclo di vita dei dati, dalla creazione alla distruzione. Allo stesso tempo, deve implementare tecniche di aggregazione e di filtraggio per ridurre l'overhead di gestione. Infatti, le tecniche di aggregazione sono utili per il ragionamento sui dati di contesto, così da ottenere informazioni coincise e di più alto livello. Le tecniche di filtraggio invece sono utili per modellare la distribuzione dei dati di contesto, così da ridurre l'overhead di gestione.

Le ricerche effettuate dall'applicazione context-aware vengono solitamente immagazzinate in alcuni oggetti, chiamati query, propagati poi all'infrastruttura. Il compito dell'infrastruttura sarà quello di restituire i risultati della ricerca al nodo chiamante e, nello specifico, all'applicazione context-aware che ha effettuato l'operazione di ricerca. Ciò avviene tramite l'analisi della query ricevuta e l'ottenimento di dati che essa soddisfa, provenienti da altri nodi presenti nell'infrastruttura. Anche i risultati verranno immagazzinati in particolari oggetti, riconoscibili dall'applicazione. Un modello che rispecchia uno scenario di questo tipo è il modello di tipo publish/subscribe, descritto nel Capitolo 2.

1.6 Sistema di distribuzione dei dati di contesto

Questo paragrafo descrive la distribuzione dei dati di contesto sia da un punto di vista locale che distribuito. Dapprima sarà presentato un modello architetturale unificato per la distribuzione dei dati di contesto allo scopo di chiarire i macro componenti e le loro interazioni. Poi, siccome il deployment distribuito condiziona profondamente l'implementazione

della distribuzione dei dati di contesto, verranno forniti alcune conoscenze utili sul network deployment. Infine, per attinenza con questa tesi, la distribuzione dei dati di contesto sarà confrontata con il modello publish/subscribe per comprendere meglio le differenze in termini di requisiti, funzionalità e servizi supportati.

1.6.1 Modello architetturale unificato

Sia l'eterogeneità che la complessità dei requisiti proposti nel Paragrafo 1.5 richiedono soluzioni complesse per la distribuzione dei dati di contesto; tali soluzioni devono distribuire le informazioni in maniera trasparente a tutte le entità interessate, monitorando costantemente le risorse disponibili e garantendo la QoC stabilita. Al crescere del sistema, l'overhead introdotto dalla distribuzione del contesto aumenta, si richiedono quindi soluzioni decentralizzate per implementare la funzione di distribuzione.

I sistemi per la distribuzione del contesto sono concepiti come architetture *data-centric* che coinvolgono tre principali attori [**Bel12**]: le sorgenti dei dati di contesto, i fruitori (o *sink*) dei dati di contesto e la funzione di distribuzione dei dati di contesto, come mostra la Figura 1.4. Il

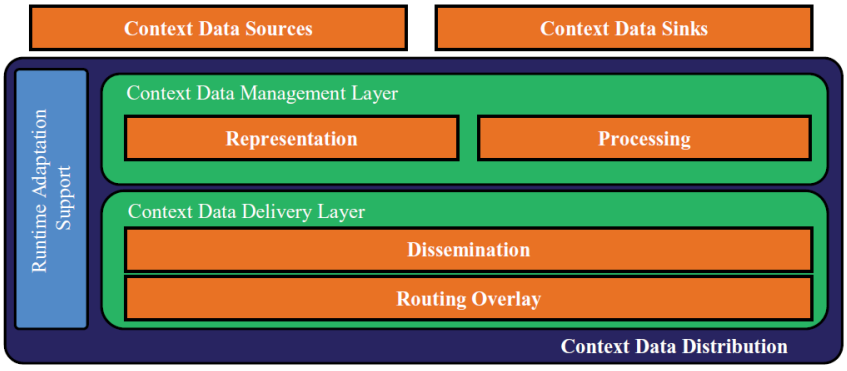

Figura 1.4: Architettura logica di un sistema di distribuzione dei dati di contesto.

Context Data Source nasconde le operazioni di accesso ai sensori e permette la pubblicazione dei dati di contesto. Il *Context Data Sink* permette a livello di servizio di esprimere le necessità sui dati di contesto utilizzando sia le query di contesto (interazione *pull-based*) che le *subscription* (interazione

push-based). Il *context data matching* è il soddisfacimento delle richieste del sink, sia query che subscription. Infine, la *context data distribution function* distribuisce i dati di contesto mediando l'interazione tra sorgenti e fruitori. La Figura 1.4 descrive inoltre l'architettura interna della funzione di distribuzione dei dati di contesto. Essa contiene tre principali funzionalità organizzate in due livelli orizzontali, *Context Data Management* e *Context Data Delivery*, e una funzionalità verticale, *Runtime Adaptation Support*, che attraversa i due livelli.

Il livello Context Data Management si occupa della gestione dei dati di contesto locali, definendo la loro rappresentazione ed esprimendo le necessità e le operazioni di elaborazione. La rappresentazione dei dati di contesto include tutti i differenti modelli e tecniche, spaziando dalle semplici e piatte coppie chiave-valore fino alle ontologie. L'elaborazione dei dati di contesto include:

- la produzione di nuova conoscenza partendo dai dati di contesto esistenti utilizzando tecniche di aggregazione;
- semplici tecniche di filtraggio per adattare la distribuzione dei dati di contesto alle risorse disponibili sul momento, così da favorire la scalabilità del sistema.

Il livello Context Data Delivery comprende sia metodi che algoritmi per consegnare i dati di contesto all'interno del sistema in maniera corretta. Tale livello implementa tutti i protocolli di disseminazione e di coordinazione richiesti per portare i dati di contesto pubblicati ai servizi context-aware interessati.

Infine, il *Runtime Adaptation Support* permette la gestione dinamica e l'adattamento degli altri livelli in accordo con le condizioni correnti di runtime, come ad esempio l'ambiente di deployment, le condizioni di una risorsa monitorata e i requisiti di QoC.

Queste tre funzionalià devono collaborare attentamente per assicurare il rispetto dei principali requisiti esposti nel Paragrafo 1.5. Contempora-

neamente, il supporto di adattamento a runtime deve considerare le risorse disponibili e le politiche di QoC per limitare e guidare le riconfigurazioni delle funzionalità ed anche per avvisare il sistema o l'utente quando le risorse correnti non sono in grado di soddisfare la QoC richiesta. Infine, per gestire il ciclo di vita dei dati di contesto, il livello Context Data Management deve considerare il tempo di generazione e il tempo di vita dei dati di contesto per consentire possibili eliminazioni di dati e, allo stesso tempo, per fornire tecniche di elaborazione , ovvero, operatori di filtraggio e di aggregazione sui dati.

1.6.2 Network deployment

Considerando che il deployment distribuito condiziona l'implementazione dei sistemi reali per la distribuzione dei dati di contesto, di seguito vengono presentati i vantaggi e gli svantaggi delle principali categorie di deployment distribuiti, già discusse nel Paragrafo 1.2.

- Modalità infrastruttura: la distribuzione dei dati di contesto sfrutta alcuni servizi raggiungibili attraverso l'infrastruttura wireless. L'utilizzo di un'infrastruttura fissa assicura un'alta disponibilità del contesto ma implica al contempo anche dei forti vincoli sull'ambiente in quanto il sistema non è in grado di lavorare in scenari privi di un'infrastruttura; inoltre, i nodi che non utilizzano la tecnologia wireless adottata dall'infrastruttura non possono far parte del sistema.

- Modalità ad-hoc: la distribuzione dei dati di contesto deve essere implementata in modo decentralizzato, mentre collegamenti ad-hoc supportano la trasmissione tra diversi nodi mobili. Quest'approccio è particolarmente adatto a scenari privi di infrastruttura fissa, come campi di battaglia e situazioni di emergenza, ma peggiorano la disponibilità dei dati e la relativa gestione.

- Reti miste: combina le due precedenti modalità cercando di ottenere il meglio da entrambe: l'alta disponibilità per i nodi in grado di comunicare con l'infrastruttura fissa, la riduzione dell'ove-

rhead per l'infrastruttura fissa sfruttando le comunicazioni ad-hoc per il reperimento dei dati e il raggiungimento dei nodi altrimenti irraggiungibili.

1.6.3 Modello publish/subscribe in ambiente mobile

In letteratura sono state proposte molte soluzioni pub/sub per ambienti mobili e alcune di esse realizzano la distribuzione di contesto [**Bel12**]. Ci sono tuttavia alcune caratteristiche differenti; analizzando i cinque requisiti evidenziati nel Paragrafo 1.5, si può concludere che i primi due sono relativi ai sistemi mobili e sono supportati da molti sistemi pub/sub, mentre i restanti tre requisiti non possono essere soddisfatti dal modello pub/sub nativo. In dettaglio:

- Il *disaccoppiamento* è una caratteristica intrinseca del modello pub/-sub.
- L'adattamento all'*eterogeneità*, intrinseca dei sistemi mobili in quanto raggruppano diversi dispositivi, è supportata anche da molti sistemi pub/sub.
- Le architetture pub/sub mobili vogliono costruire delle primitive di comunicazione per l'intero sistema che tipicamente non permettono l'applicazione di *scope di visibilità* per i dati di contesto.
- Poi, i sistemi pub/sub di solito non considerano la consegna basata su *garanzie di qualità*. Inoltre, la distribuzione dei dati di contesto deve considerare sia dati che subscription incerti, mentre le subscription tipiche dei sistemi pub/sub considerano solo match subscription/dato perfetti.
- Infine, i sistemi pub/sub non gestiscono esplicitamente il *ciclo di vita dei dati*. Anche se i sistemi pub/sub forniscono alcune soluzioni per la rimozione di dati/messaggi, non mettono a disposizione operazioni più complesse quali, ad esempio, l'aggregazione di dati/messaggi.

In conclusione, sebbene la distribuzione dei dati di contesto mostra diverse somiglianze con i sistemi pub/sub, questi ultimi non sono nativamente in grado di soddisfare a pieno tutti i requisiti della distribuzione dei dati di contesto.

1.7 Obiettivo della tesi

Partendo dal dato di fatto che in un sistema context-aware le query emesse da nodi che si trovano nella stessa area geografica presentano di solito similarità e coperture parziali o totali, l'obiettivo di questa tesi è quello di evidenziare e sfruttare tali similarità per modulare opportunamente il carico di lavoro dei singoli nodi ed il traffico dei dati di contesto generato in risposta alle richieste.

Nella prima fase del lavoro di tesi è stata analizzata un'infrastruttura di distribuzione dei dati di contesto, sviluppata dall'Università di Bologna sotto il nome di SALES, descritta nel Capitolo 3. Dopo aver compreso l'architettura generale del sistema, è stato necessario analizzare in maniera approfondita gli aspetti legati alla struttura delle query e dei dati di contesto, alla loro creazione, gestione delle copie in locale, gestione della diffusione e all'implementazione della procedura di match tra query e dato di contesto.

Poi sono stati indagati i principali sforzi presenti in letteratura che sfruttano le similarità tra le query per migliorare l'efficienza dei sistemi, come descritto nel Capitolo 4. Tali studi riguardano principalmente il modello pub/sub, per cui si è ritenuto opportuno darne una presentazione nel Capitolo 2.

Nella seconda fase del lavoro di tesi, le idee ed i concetti presentati nel Capitolo 4 sono stati adattati per essere compatibili con i sistemi context-aware ed in particolar modo con l'infrastruttura SALES. Poi, è stato progettato ed implementato un componente software in grado di adattare il

proprio comportamento in base alle risorse utilizzate dal dispositivo sul quale esegue. Tale componente è da considerarsi il nucleo della soluzione. Inoltre, è stata realizzata ed integrata in SALES una soluzione in grado di sfruttare il precedente componente software in maniera tale da interagire correttamente con esso. Infine, sono stati eseguiti opportuni test su differenti deployment per verificare la correttezza del comportamento e le performance della soluzione.

Capitolo 2

Il modello Publish/Subscribe

Si è ritenuto opportuno introdurre un capitolo descrittivo del modello Publish/Subscribe poiché in letteratura sono presenti molti studi applicati ad esso con obiettivi simili a quelli della presente tesi.

Internet ha notevolmente cambiato le dimensioni dei sistemi distribuiti, coinvolgendo migliaia di entità potenzialmente distribuite in tutto il mondo, le quali posizioni e comportamenti potrebbero variare durante il tempo di vita del sistema [**Eug03**]. Queste esigenze spingono a creare dei sistemi e modelli di comunicazione piú flessibili, che riflettano la natura dinamica e disaccoppiata delle applicazioni. Le comunicazioni sincrone point-to-point tendono a generare applicazioni statiche e rigide, rendendo difficoltoso lo sviluppo di applicazioni dinamiche su larga scala. Per ridurre la difficoltà di progettazione di tali applicazioni è necessario interporre tra le diverse entità un'infrastruttura di comunicazione flessibile e scarsamente accoppiata.

Questo capitolo è strutturato in nei seguenti paragrafi: il Paragrafo 2.1 mostra lo schema di interazione base e le sue proprietà, il Paragrafo 2.2 caratterizza le principali varianti del modello publish/subscribe ed infine il Paragrafo 2.3 descrive alcuni aspetti implementativi dello schema publish/subscribe in merito agli eventi, il canale di comunicazione e la qualità di servizio.

2.1 Schema di interazione

Il modello publish/subscribe prevede l'interazione tra due tipologie principali di soggetti: i produttori delle informazioni, detti *publisher*, e i consumatori delle informazioni, detti *subscriber*. Il paradigma d'interazione di tale modello consente ai subscriber di esprimere i loro interessi e di ricevere successivamente qualsiasi informazione a riguardo, indipendentemente da chi l'abbia prodotta. L'informazione è tipicamente chiamata con il nome di *evento*, mentre la sua consegna è identificata con il nome di *notifica*.

Il modello base d'interazione publish/subscribe, mostrato in Figura 2.1, sfrutta un servizio di notifica degli eventi che fornisce la memorizzazione e la gestione delle registrazioni e la consegna efficiente degli eventi. Tale servizio funge da intermediario neutrale tra i publisher, agendo come

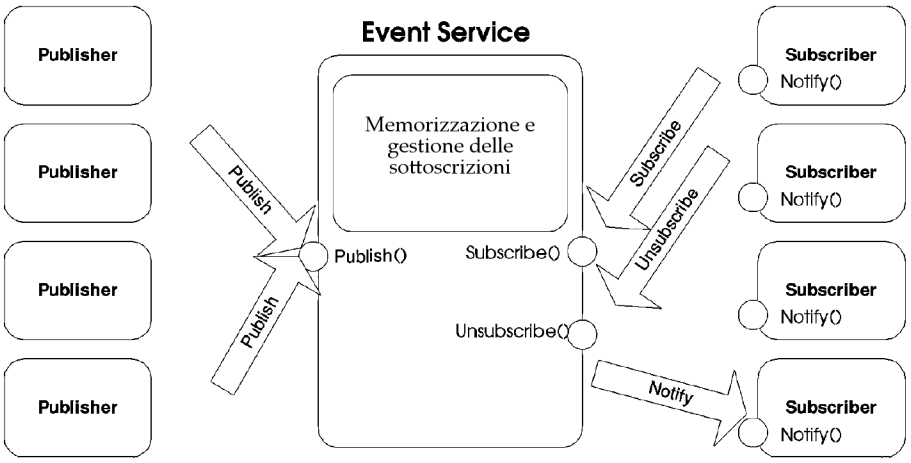

Figura 2.1: Un esempio di sistema publish/subscribe.

produttori di eventi, e i subscriber, agendo come consumatori di eventi. I subscriber registrano il loro interesse su gli eventi attraverso una chiamata al metodo `subscribe()`, definito presso il gestore degli eventi, senza conoscere chi effettivamente li produce. L'informazione sulla registrazione resta memorizzata sul gestore degli eventi e non viene inoltrata ai publisher. L'operazione simmetrica `unsubscribe()` termina la registrazione e viene invocata dal subscriber quando non vuole più ricevere nessuna o alcune informazioni.

Per generare un evento i publisher invocano il metodo `publish()`, definito anch'esso presso il gestore degli eventi. Quest'ultimo ha il compito di propagare l'evento a tutti i subscriber interessati, comportandosi di fatto come un proxy. I publisher hanno inoltre la capacità di comunicare la natura dei loro eventi futuri attraverso il metodo `advertise()`. Questa operazione ha un duplice vantaggio: da un lato consente al gestore di eventi di adattarsi al flusso di eventi atteso, dall'altro permette ai subscriber di sapere quando diventa disponibile un nuovo tipo di informazione.

Il gestore degli eventi introduce il disaccoppiamento tra publisher e subscriber che può essere suddiviso nelle seguenti tre dimensioni:

- *Spazio*: non è necessaria la conoscenza reciproca delle parti che interagiscono. I publisher pubblicano gli eventi sul gestore di eventi, il quale si occupa dell'inoltro verso i subscriber. Tipicamente i publisher non hanno riferimenti ai subscriber e neppure conoscono quanti di essi stanno partecipando all'interazione. Allo stesso modo, i subscriber non hanno riferimenti ai publisher e neppure conoscono da quanti di essi stanno ricevendo informazioni. La 2.2 mostra il disaccoppiamento spaziale.

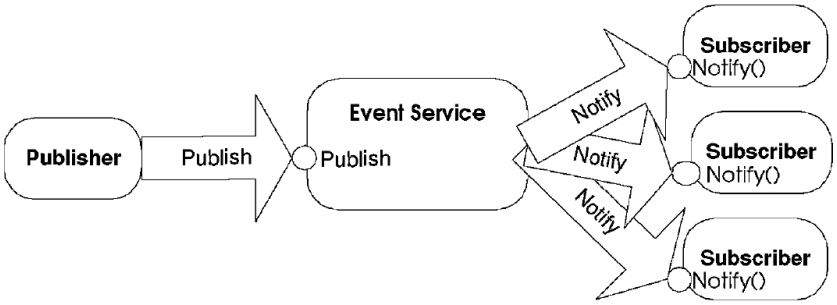

Figura 2.2: Disaccoppiamento spaziale.

- *Tempo*: non è necessaria la partecipazione attiva delle parti che interagiscono nello stesso istante temporale. In particolare, il publisher potrebbe pubblicare degli eventi quando il subscriber è disconnesso, mentre il subscriber potrebbe essere notificato dell'occorren-

za di alcuni eventi mentre il publisher originale è disconnesso. Il disaccoppiamento temporale è mostrato in Figura 2.3.

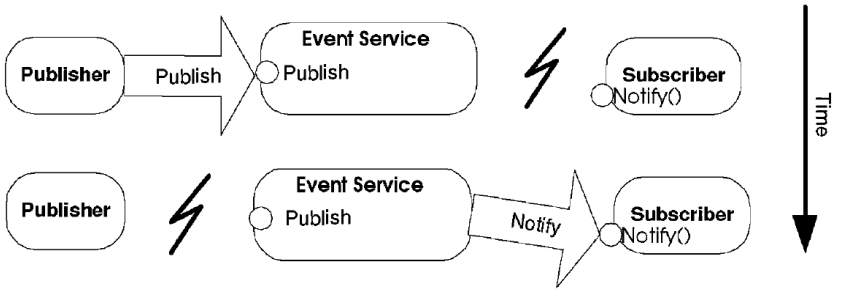

Figura 2.3: Disaccoppiamento temporale.

- *Sincronizzazione*: i publisher non si bloccano durante la produzione di eventi e i subscriber possono essere notificati in maniera asincrona dell'occorrenza di un evento mentre eseguono delle attività concorrenti. Tale notifica prende il nome di *callback*. La produzione e il consumo di eventi non avvengono nel principale flusso di controllo del publisher e del subscriber, perciò non avvengono in maniera sincrona. La Figura 2.4 mostra il disaccoppiamento di sincronizzazione.

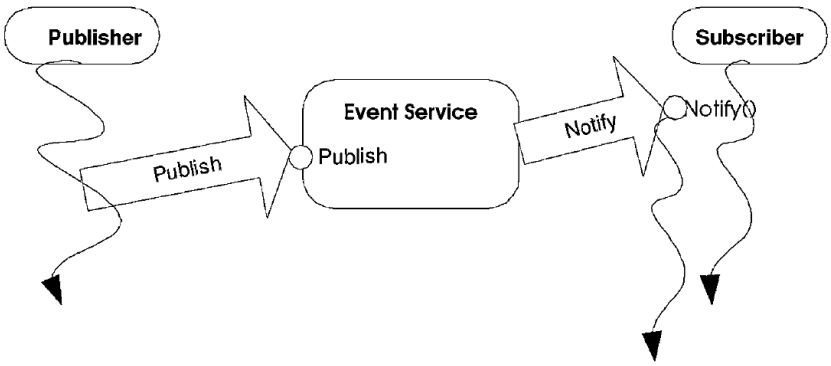

Figura 2.4: Disaccoppiamento in sincronizzazione.

Disaccoppiando la produzione di informazioni dal loro consumo si incrementa la scalabilità, rimuovendo tutte le dipendenze esplicite tra i partecipanti che interagiscono. Tale rimozione riduce fortemente il coordinamento e perciò la sincronizzazione tra le differenti entità; ciò rende l'infrastruttura di comunicazione adatta ad ambienti distribuiti, asincroni per loro natura, come gli ambienti mobili [**Hua01**].

2.2 Aspetti implementativi

Questo paragrafo introduce alcuni aspetti implementativi dello schema publish/subscribe molto complessi e mostra come tali sfide siano state affrontate nei prototipi e nei sistemi più utilizzati. L'attenzione sarà rivolta sui tre principali aspetti del modello publish/subscribe: l'evento, il canale di comunicazione e la qualità di servizio. Inoltre saranno esposti i trade-off dei differenti approcci, in termini di flessibilità, affidabilità, scalabilità e performance.

2.2.1 Evento

Gli eventi possono essere di due tipi differenti: messaggio o invocazione. Nel primo caso gli eventi sono consegnati ad un subscriber attraverso una singola operazione generale (ad esempio `notify()`), mentre nel secondo caso gli eventi innescano l'esecuzione di specifiche operazioni da parte del subscriber.

Messaggi

A basso livello, ogni dato che va sulla rete è un pacchetto. Nella maggior parte dei sistemi le notifiche degli eventi avvengono sotto forma di messaggi creati esplicitamente dall'applicazione. Tipicamente i messaggi contengono una intestazione, o *header*, contenente informazioni specifiche sul messaggio in un formato generico ed un corpo, o *payload*, contenente le informazioni vere e proprie. Di solito i campi dell'intestazione includono l'identificativo del messaggio, il mittente e la priorità (o il tempo di scadenza), la quale può essere utilizzata dal sistema o solamente dal consumatore. Alcuni sistemi non fanno nessuna assunzione sul tipo del corpo del messaggio e lo trattano come un array opaco di byte. Altri sistemi, invece, forniscono un insieme di tipi di messaggi, come ad esempio messaggi testuali oppure messaggi XML. Infine, alcuni sistemi forniscono messaggi autodescrittivi. Distributed Asynchronuos Collections (DAC) [**Eug00**] e Java Message Service [**Hap02**] supportano anche i messaggi sotto forma

di oggetto, dove l'evento può essere qualsiasi oggetto Java serializzabile. Nella maggior parte dei casi, i messaggi sono visti come un documento con molti campi.

Invocazioni

A livello più alto si fa differenza tra *invocazioni* e *messaggi*. Una invocazione è diretta ad uno specifico tipo di oggetto ed ha una semantica ben definita. Il sistema assicura che tutti i consumatori hanno un'interfaccia corretta per l'elaborazione dell'invocazione. L'interfaccia agisce da contratto vincolante tra chi invoca e chi è invocato. Mentre alcuni sistemi considerano i valori di ritorno dell'invocazione, i modelli publish/subscribe di COM+ [**Ses97**] e l'Event Service di CORBA [**Omg01**] considerano solo invocazioni one-way. I sistemi che offrono un'interazione basata sull'invocazione con differenti semantiche e svariati schemi di indirizzamento sono chiamati sistemi di *messaggistica*. Essi introducono della logica addizionale in cima al sistema publish/subscribe, trasformando i messaggi di basso livello in invocazioni di metodo per i subscriber. I produttori invocano le operazioni su oggetti intermediari, ad esempio l'*event channel*, che esibiscono la stessa interfaccia del consumatore ed inoltrano gli eventi a tutti i consumatori registrati. COM+ fornisce anche una forma di filtraggio basato sul contenuto, offrendo la possibilità di specificare i valori degli argomenti dell'invocazione per restringere il numero delle potenziali invocazioni.

2.2.2 Canale di comunicazione

La trasmissione dei dati dai produttori ai consumatori è compito dell'infrastruttura. Il mezzo di comunicazione può essere classificato in accordo con le sue caratteristiche come l'architettura o le garanzie che fornisce ai dati, quali persistenza o affidabilità.

Architettura

Il ruolo dei sistemi publish/subscribe consente lo scambio di eventi in maniera asincrona tra produttori e consumatori. L'asincronia può essere implementata con l'invio dei messaggi dai produttori ad una particolare entità che li memorizza e li inoltra ai consumatori su richiesta. Questo approccio viene identificato con il nome di architettura *centralizzata*, per la presenza dell'entità centralizzata che memorizza e inoltra i messaggi. Tale scelta è adottata dai sistemi di accodamento dei messaggi come IBM MQSeries [**Lew99**] e Oracle Advanced Queuing [**Ora02**], i quali si basano entrambe sull'utilizzo di un database centralizzato. Le applicazioni basate su tali sistemi hanno dei requisiti forti in termini di affidabilità, consistenza dei dati o supporto per le transizioni, ma non necessitano di un'elevata capacità di trasmissione dei dati. Esempi di tali applicazioni sono costituiti dal commercio elettronico o da applicazioni bancarie.

L'asincronia può essere implementata anche tramite l'utilizzo di primitive di comunicazione intelligenti che implementano meccanismi di memorizzazione e di inoltro, sia nei processi del produttore che in quelli del consumatore; in tal modo la comunicazione sembra asincrona e anonima alle applicazioni e non necessita di un'entità intermediaria. Quest'approccio viene chiamato con il nome di architettura *distribuita* perché non c'è nessun punto di centralizzazione nel sistema. TIBCO Rendezvous [**Tib99**] usa un approccio di questo tipo nel quale nessun processo agisce come collo di bottiglia o singolo punto di fallimento. Tali architetture sono adatte per la consegna veloce ed efficiente di dati transienti, requisito indispensabile per applicazioni quali borsa o broadcasting multimediale.

Un approccio intermedio consiste nell'implementazione del servizio di notifica degli eventi come rete distribuita di server. A differenza dell'approccio completamente decentralizzato, esso semplifica i processi partecipanti utilizzando dei server dedicati per eseguire i protocolli richiesti per la persistenza, l'affidabilità e l'*high-availability*, così come il filtraggio basato sul contenuto e il routing. Ci sono differenti tipologie per tali

server:

- In Jedi [**Cug01**] c'è l'*event dispatcher* che ha una struttura gerarchica dove i clienti possono connettersi con ogni nodo. Le registrazioni sono propagate verso la radice dell'albero dei server. Tuttavia, tali topologie gerarchiche tendono a caricare pesantemente la radice dei server e il fallimento di un server potrebbe disconnettere l'intero sotto-albero.

- In Gryphon [**Ban99**] c'è un grafo riassuntivo degli interessi dei subscriber che è sovrapposto al grafo *message broker* per evitare ridondanze nella corrispondenza registrazioni/eventi.

- In Siena [**Car00**] si usa l'inoltro di registrazioni e inserzioni per impostare il cammino delle notifiche. Gli *event server* tengono traccia delle informazioni utili per verificare in maniera efficiente la corrispondenza registrazioni/eventi.

Diffusione dei dati

L'effettiva trasmissione dei dati può avvenire in modi differenti. In particolare, i dati possono essere inviati utilizzando primitive di comunicazioni point-to-point oppure funzionalità hardware per il multicast come IP multicast [**Dee**]. La scelta dei meccanismi di comunicazione dipende da fattori come l'ambiente target e l'architettura del sistema.

Di solito gli approcci centralizzati, come certi sistemi di accodamento dei messaggi, utilizzano primitive di comunicazione point-to-point tra produttori/consumatori e il broker centralizzato. Come già detto, tali sistemi prediligono garanzie forti a discapito del throghput e della scalabilità. I sistemi publish/subscribe basati sui topic possono sicuramente beneficiare dell'elevato numero di studi sui gruppi di comunicazione [**Pow96**] e sui protocolli di diffusione degli eventi ai subscriber. Per assicurare un elevato throughput, vengono comunemente impiegati il multicast dell'Internet Protocol (IP) oppure altri protocolli affidabili di multicast.

A tutt'oggi risulta difficile implementare un multicast efficiente di eventi nei sistemi publish/subscribe basati sul contenuto. Entrambi i sistemi Gryphon e Siena utilizzano degli algoritmi che consegnano gli eventi ad una rete logica di server, in maniera tale che un evento è propagato solo ai server che gestiscono i subscriber interessati all'evento stesso. La performance di tali sistemi basati sulla diffusione è fortemente limitata dal costo del filtraggio degli eventi su ognuno dei server, il quale dipende direttamente dal numero di registrazioni presenti nel sistema. Da tali limitazioni sono nati algoritmi scalabili ed altamente efficienti per il filtraggio dei dati in sistemi publish/subscribe. È stato inoltre studiato il problema dell'aggregazione delle registrazioni **[Cha02]** per incrementare la velocità di filtraggio in ogni server. Indipendentemente dalle tecniche di filtraggio, il routing selettivo degli eventi, intrinseco del modello publish/subscribe basato sul contenuto, rende difficoltoso lo sfruttamento delle primitive di multicast a livello di rete.

2.2.3 Qualità di servizio

Le garanzie fornite per ogni messaggio dal canale di comunicazione variano molto tra i diversi sistemi. Le più comuni proprietà riguardanti la qualità di servizio per i sistemi publish/subscribe sono: persistenza, garanzie transazionali e priorità.

Persistenza

Nei sistemi basati su RPC, un'invocazione di metodo è per definizione un evento transiente. Il tempo di vita di un'invocazione remota è breve e, se il chiamante non riceve alcuna risposta entro un dato periodo di tempo, si potrebbe effettuare nuovamente la richiesta. Nei sistemi publish/subscribe la situazione è differente: i messaggi inviati potrebbero non generare alcuna risposta e potrebbero essere processati ore dopo il loro invio. Le parti comunicanti non controllano come i messaggi vengono trasmessi e quando sono processati; perciò, il sistema di messaggistica deve fornire garanzie non solo in termini di affidabilità, ma anche in termini di durata

delle informazioni. Non è sufficiente sapere che un messaggio ha raggiunto il sistema di messaggistica posto tra i produttori e i consumatori, ma bisogna avere garanzie che un messaggio non andrà perso a causa di fallimenti del sistema di messaggistica.

Tipicamente la persistenza è garantita da ogni sistema publish/subscribe con un'architettura centralizzata; l'entità centrale memorizza i messaggi consentendone la ricezione anche per un subscriber difettoso. I sistemi di accodamento di messaggi, come ad esempio IBM MQSeries [**Lew99**] e Oracle Advanced Queuing [**Ora02**], offrono un supporto per la persistenza mediante un database sottostante. Al contrario, i sistemi publish/-subscribe distribuiti non offrono un supporto per la persistenza in quanto i messaggi vengono direttamente inviati dal produttore a tutti i subscriber. Se il produttore non mantiene una copia di ogni messaggio inviato, un subscriber difettoso potrebbe non essere in grado di recuperare i messaggi perduti durante la fase di recovery. TIBCO Rendezvous [**Tib99**] offre un approccio misto nel quale i processi si mettono in ascolto di specifici soggetti, memorizzano i messaggi su memoria persistente e rinviano quelli persi ai subscriber. Il Cambridge Event Architecture [**Bac00**] fornisce un repository di eventi potenzialmente distribuito per la memorizzazione e il recupero efficace di eventi e il rinvio di sequenze di eventi memorizzati.

Priorità

Oltre alla persistenza, anche l'assegnazione delle priorità è una qualità di servizio che viene garantita da diversi sistemi di messaggistica. Tali sistemi si basano sul fatto che i subscriber possono trarre vantaggio dall'elaborazione ordinata dei messaggi in base alle loro priorità. Per esempio, un messaggio real time potrebbe richiedere una reazione immediata e andrebbe elaborato prima degli altri messaggi; si pensi ad una notifica di fallimento.

Le priorità condizionano i messaggi che sono *in transito*, ovvero quei messaggi che non sono ancora stati elaborati. Durante l'esecuzione le

priorità sono gestite dallo scheduler delle applicazioni e non dal sistema di messaggistica. Più in dettaglio, questo implica che due subscriber in ascolto per lo stesso topic, potrebbero processare i messaggi in ordine diverso perché elaborano i messaggi a velocità diversa, anche se i canali di comunicazione sono first-in/first-out (FIFO). Le priorità andrebbero considerate come una qualità di servizio best-effort, ovvero senza garanzie, differentemente dalla persistenza.

Molti sistemi di messaggistica publish/subscribe, centralizzati o distribuiti, consentono l'utilizzo delle priorità, ma non vi è uno standard che definisce il numero di priorità o il modo in cui applicarle. Tutti i sistemi IBM MQSeries [**Lew99**], Oracle Advanced Queuing [**Ora02**], TIBCO Rendezvous [**Tib99**] e la specifica JMS [**Hap02**] supportano le priorità.

Transazioni

Le transazioni sono spesso usate per raggruppare operazioni multiple in blocchi atomici che devono essere interamente eseguiti oppure non eseguiti affatto, seguendo la semantica *tutto o niente*. Nei sistemi di messaggistica, le transazioni sono usate per raggruppare messaggi in unità atomiche: o l'intera sequenza di messaggi deve essere inviata/ricevuta tutta oppure nessun messaggio deve essere inviato/ricevuto. Per esempio, un produttore che fallisce durante l'invio di diversi messaggi correlati semanticamente, potrebbe non volere che i consumatori vedano una sequenza parziale, quindi inconsistente, di tali messaggi. In maniera simile, un'applicazione critica potrebbe consumare uno o più messaggi, elaborarli e, solo poi, effettuare il *commit* della transazione. Se il consumatore dovesse fallire prima di effettuare il commit, tutti i messaggi sarebbero ancora disponibili per la rielaborazione dopo la fase di recovery.

Per la loro integrazione stretta con i database, IBM MQSeries [**Lew99**] e Oracle Advanced Queuing [**Ora02**] forniscono un vasto range di meccanismi transazionali. Anche JMS [**Hap02**] e TIBCO Rendezvous [**Tib99**] forniscono un supporto alle transazioni per il raggruppamento di messaggi

nel contesto di una singola sessione. JavaSpaces [**Fre99**] fornisce dei meccanismi transazionali leggeri per garantire l'atomicità della produzione e del consumo degli eventi. Un evento pubblicato in JavaSpace nel contesto di una transazione, non è visibile all'esterno della transazione finché non viene effettuato il commit. Alla stessa maniera, un evento consumato non viene rimosso dallo JavaSpace finché non termina il commit della transazione. All'interno di una stessa transazione possono essere prodotti e consumati molti eventi.

Affidabilità

L'affidabilità è un'importante caratteristica dei sistemi distribuiti. È spesso necessario avere garanzie forti sulla consegna affidabile delle informazioni verso una o più entità distribuite. A causa della scarsa sincronizzazione tra produttori e consumatori di informazioni, l'implementazione dell'affidabilità nella propagazione degli eventi (garanzia di consegna) risulta complessa.

I sistemi publish/subscribe centralizzati utilizzano spesso i canali di comunicazione point-to-point affidabili per mettere in comunicazione i publisher con i subscriber e per mantenere delle copie degli eventi su una memoria stabile. Gli eventi sono perciò consegnati in modo affidabile a tutti i subscriber, anche se un fallimento del broker di eventi centralizzato potrebbe ritardare la consegna.

I sistemi basati su una rete di broker di eventi distribuiti utilizzano spesso protocolli affidabili per propagare gli eventi a tutti o ad un sottoinsieme di broker. I protocolli basati sulle comunicazioni di gruppo (come [**Pow96**]) e il multicast affidabile a livello applicazione (come [**Flo97**], [**Hol95**], [**Lin96**], [**Cas02**], [**Ban02**], [**Rat01**] e [**Zhu01**]) sono dei buoni candidati in quanto elastici al fallimento di alcuni broker. Sia publisher che subscriber comunicano con il broker a loro più vicino utilizzando canali di comunicazione point-to-point.

Infine, anche i sistemi che permettono una comunicazione diretta tra publisher e subscriber (come TIBCO Rendezvous [**Tib99**]) utilizzano un

protocollo leggero per il multicast affidabile. Siccome gli eventi spesso non sono mantenuti nel sistema, la garanzia di consegna dev'essere implementata utilizzando dei processi dedicati che memorizzano gli eventi e li rinviano ai subscriber che hanno subito un fallimento o erano semplicemente disconnessi (disaccoppiamento temporale).

2.3 Varianti del modello Publish/Subscribe

Nella maggior parte dei casi i subscriber non sono interessati a tutti gli eventi, ma solo ad alcuni in particolare o a quelli che rispettano un particolare schema. Esistono diversi modi per specificare gli eventi di interesse, ognuno dei quali definisce un preciso schema di registrazione. In questa sezione saranno confrontati gli schemi più usati: *topic-based*, *content-based* ed il successivo *type-based*.

2.3.1 Topic-based

Il primo schema publish/subscribe si basa sulla nozione di *topic*, ovvero argomento o soggetto, ed è comunemente implementato da molti sistemi industriali: si estende la nozione di canale, usato per raggruppare i nodi comunicanti, con metodi che consentono di caratterizzare e classificare il contenuto degli eventi. I partecipanti possono pubblicare eventi e/o registrarsi presso singoli topic, i quali sono identificati da parole chiave. Il topic è fortemente correlato alla nozione di *gruppo*, come definito nel contesto di gruppo di comunicazione [**Pow96**], ed è spesso usato per la replicazione [**Bir93**]. Questa somiglianza non deve stupire poiché alcuni dei primi sistemi publish/subscribe si basavano su Isis [**Bir90**], ovvero un insieme di strumenti per la comunicazione di gruppo. In particolare, registrarsi per un topic T può essere associato all'azione di diventare membro di un gruppo T; allo stesso modo, pubblicare un evento per il topic T significa trasmetterlo a tutti i membri del gruppo T. Sebbene gruppi e topic sono astrazioni simili, vengono tipicamente associati a differenti domini applicativi: i gruppi sono usati per mantenere una consistenza forte tra le

repliche di un componente critico in una *local area network* (LAN), mentre i topic sono usati per modellare le interazioni distribuite su larga scala.

In pratica, i sistemi publish/subscribe basati sui topic introducono un'astrazione di programmazione la quale mappa ogni topic su un distinto canale di comunicazione. Le interfacce presentate dal topic sono molto simili a quelle del servizio di notifica degli eventi presentato nel Paragrafo 2.1 e il nome del topic è tipicamente definito come parametro di inizializzazione. Ogni topic , identificato con un nome univoco, è visto come un servizio di gestione degli eventi di se stesso e la sua interfaccia mostra le operazioni di `publish()` e di `subscribe()`.

L'astrazione di topic rafforza l'interoperabilità della piattaforma in quanto si basa solo su stringhe di testo come chiave per la suddivisione dello spazio di eventi. Diversi sistemi hanno effettuato delle aggiunte allo schema publish/subscribe; la più utile tra tutte è stata l'utilizzo delle *gerarchie* per coordinare i topic. Mentre i sistemi basati sui gruppi offrono un *indirizzamento piatto*, dove i gruppi rappresentano spazi disconnessi di eventi, tutti i moderni sistemi basati sui topic offrono una forma di *indirizzamento gerarchico*, consentendo ai programmatori di organizzare i topic in accordo con le relazioni di contenimento/isolamento. Una registrazione fatta presso un nodo della gerarchia scatena implicitamente la registrazione presso tutti i sotto-topic di quel nodo. Spesso i nomi dei topic sono rappresentati attraverso una notazione *URL-like* ed è consentito l'uso di nomi contenenti *wildcard*, ovvero caratteri jolly che possono assumere qualsiasi valore. In tal modo si permette la registrazione e la pubblicazione su particolari topic, i quali nomi si abbinano ad un insieme di parole chiave come un intero sotto-albero o un livello specifico nella gerarchia. Si supponga ad esempio di avere un'applicazione che consente di scambiare informazioni sulle quotazioni azionarie e che tali siano classificate in gerarchia per `Nazione/Luogo/Nome`; un subscriber potrà esprimere il suo interesse verso una specifica azione (`UK/London/FT-SE 100`), un particolare insieme di azioni utilizzando la wildcard `*` (`UK/London/FT-SE*`) oppure un'intera sotto-categoria di azioni (`UK/London`).

La Figura 2.5 mostra un esempio delle quotazioni finanziarie da distribuire ad un certo numero di broker interessati. Come primo passo, biso-

```
public class StockQuote implements Serializable {
  public String id, company, trader;
  public float price;
  public int amount;
}
public class StockQuoteSubscriber implements Subscriber {
  public void notify(Object o) {
    if (((StockQuote)o).company == 'TELCO' && ((StockQuote)o).price < 100)
      buy();
  }
}
// ...
Topic quotes = EventService.connect("/LondonStockMarket/Stock/StockQuotes");
Subscriber sub = new StockQuoteSubscriber();
quotes.subscribe(sub);
```

Figura 2.5: Esempio di codice per il modello basato sui topic.

gna acquistare le azioni pubblicizzate da eventi di quotazione delle azioni. Tali eventi sono descritti da 5 attributi: un identificatore globale, il nome della società, il prezzo, il numero di azioni e l'identificato dell'operatore di borsa. In Figura 2.6 viene mostrata l'interazione tra publisher e subscriber.

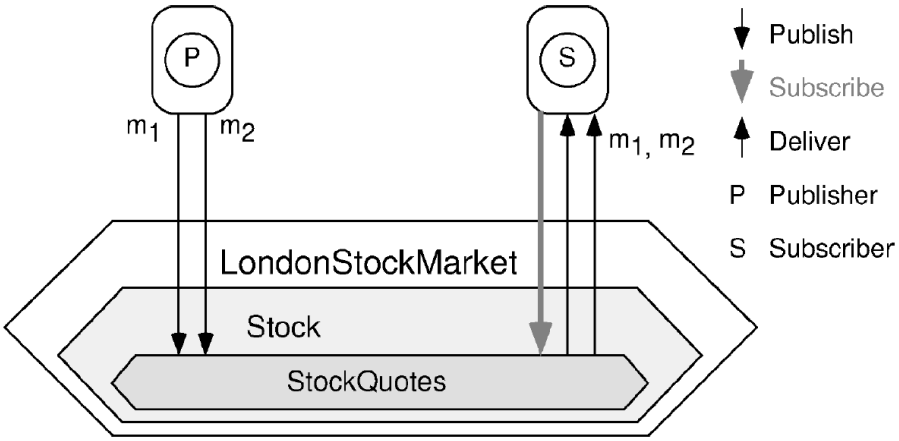

Figura 2.6: L'interazione nel modello basato sui topic.

2.3.2 Content-based

Nonostante i miglioramenti quali le funzioni di indirizzamento gerarchico e le *wildcard*, il modello publish/subscribe basato sui topic è statico ed ha un'espressività limitata. La variante basata sul contenuto, o *content-based*, perfeziona i topic introducendo uno schema di registrazione basato sull'effettivo contenuto degli eventi considerati. Gli eventi non sono più classificati secondo alcuni criteri esterni predefiniti, come ad esempio il nome del topic, ma in base alle proprietà degli eventi stessi. Tali proprietà possono essere attributi interni alle strutture dati degli eventi (come in Gryphon [**Ban99**], Siena [**Car00**], Elvin [**Seg00**] e Jedi [**Cug01**]) o afferenti ai meta-dati associati agli eventi (come in Java Message Service [**Hap02**]).

I subscriber si registrano per taluni eventi specificando i filtri mediante un linguaggio di registrazione. I filtri sono espressi nella forma di coppie *chiave-valore* sulle proprietà e dagli operatori di confronto ($<$, $\leq$, $=$, $\geq$, $>$), i quali selezionano degli eventi validi. I filtri definiscono i vincoli che, logicamente combinati (and, or, etc...), formano *pattern di registrazione* complessi. Alcuni sistemi, come il Cambridge Event Architecture (CEA) [**Bac00**], considerano anche la *correlazione degli eventi*: i partecipanti possono registrare combinazioni logiche di eventi elementari per essere notificati solo quando accade l'intero evento composto. Il pattern di registrazione è usato per identificare gli eventi di interesse di un particolare subscriber e, conseguentemente, per propagare gli eventi. Il servizio di notifica degli eventi fornisce una variante dell'operazione di registrazione `subscribe()`, introducendo un argomento che identifica il pattern di registrazione. Esistono diversi modi per rappresentare tale pattern:

- *String*: è il metodo più utilizzato e prevede che il filtro sia in accordo con una grammatica di registrazione, come ad esempio SQL, Default Filter Constraint Language di OMG, XPath o altri linguaggi proprietari.
- *Oggetto template*: all'atto della registrazione il subscriber fornisce un oggetto *t* con il quale comunica il suo interesse verso tutti gli eventi

dello stesso tipo di t e i quali attributi corrispondono a quelli contenuti in t. JavaSpaces [**Fre99**] adotta un approccio basato sugli oggetti template.

- *Codice eseguibile*: i subscriber forniscono un oggetto predicato in grado di filtrare gli eventi a tempo di esecuzione. Spesso l'implementazione di tali oggetti viene effettuata dallo sviluppatore dell'applicazione. Nella pratica tale modalità è poco utilizzata in quanto i filtri risultano molto difficili da ottimizzare.

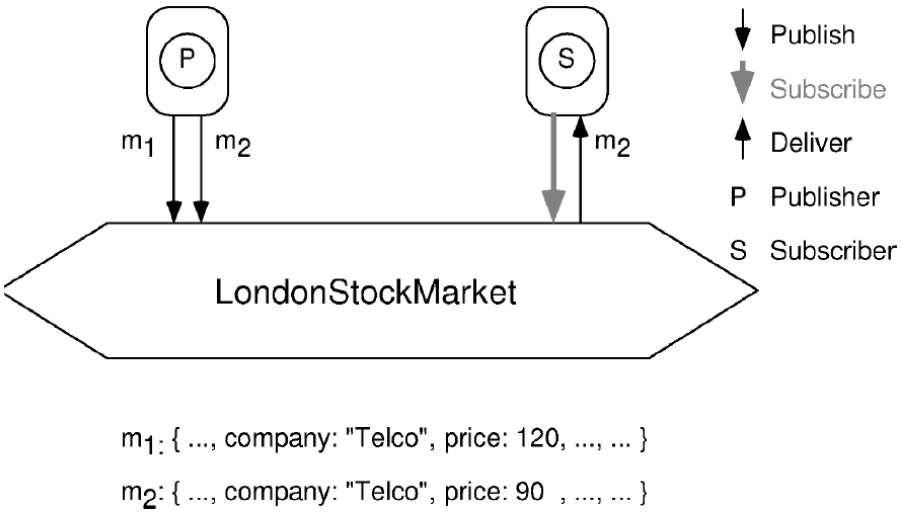

Figura 2.7: L'interazione nel modello basato sul contenuto.

La Figura 2.7 mostra l'utilizzo dei filtri basati sulle stringhe di testo. L'esempio evidenzia come uno schema basato sul contenuto impone una granularità più fine rispetto allo schema statico basato sui topic. Per avere la stessa funzionalità con gli argomenti, il subscriber dovrebbe filtrare gli eventi irrilevanti oppure i topic dovrebbero essere suddivisi in sotto-topic, uno per ogni società e ricorsivamente molti sotto-topic per differenti "categorie" di prezzo. Il primo approccio porta ad un uso inefficiente della banda dati, mentre il secondo approccio genera un elevato numero di argomenti ed un rischio di eventi ridondanti.

2.3.3 Type-based

Tipicamente gli argomenti raggruppano eventi con caratteristiche in comune per quanto riguarda il contenuto e la struttura. Questa osservazione ha portato alla sostituzione del modello di classificazione degli argomenti basato sui nomi con uno schema che filtra gli eventi in base al loro tipo. In altri termini, la nozione di *categoria* dell'evento è direttamente associata a quella di *tipo* dell'evento, rendendo possibile un'integrazione più stretta del linguaggio e dell'infrastruttura. Inoltre, la certezza del tipo viene assicurata a tempo di compilazione attraverso l'interfaccia dell'evento corrispondente. Tale schema si differenzia da quello dell'oggetto template citato precedentemente in quanto non considera il tipo di evento come una proprietà dinamica e non necessita quindi di cast espliciti.

La Figura 2.8 mostra un esempio dove gli eventi sono suddivisi in due tipi differenti: *quotare* un'azione e *richiedere* un'azione. I broker utilizzano le richieste per esprimere il loro interesse nell'acquisto di azioni; a differenza delle quotazioni, una richiesta ha un range di prezzi possibili. È possibile usare i sotto-tipi per registrarsi sia presso le quotazioni di azioni che presso le richieste.

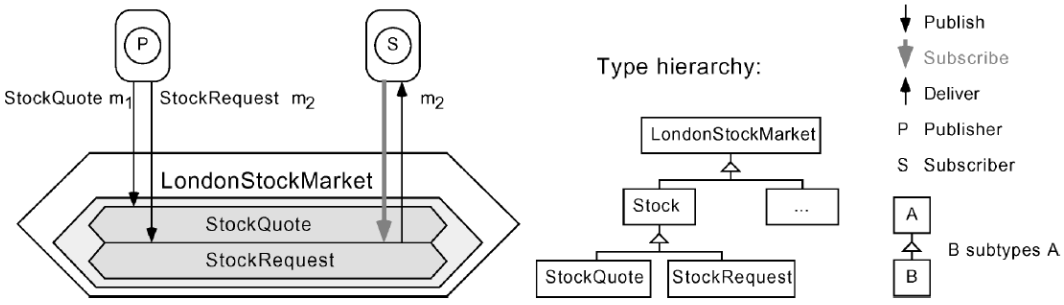

Figura 2.8: L'interazione nel modello basato sui tipi.

È importante osservare che il modello publish/subscribe basato sui tipi porta ad una naturale descrizione del filtraggio basato sul contenuto, effettuato attraverso le proprietà pubbliche del tipo di evento considerato e assicurando l'incapsulamento degli eventi.

2.3.4 Confronti

Esistono molte varianti di progettazione dei sistemi publish/subscribe, ognuna delle quali offre un differente grado di espressività e di performance. Il modello publish/subscribe basato sui topic è abbastanza statico e primitivo, ma può essere implementato in maniera molto efficiente. D'altra parte, il modello publish/subscribe basato sul contenuto è altamente espressivo, ma richiede protocolli sofisticati che introducono un elevato overhead a tempo di esecuzione. Proprio a causa di questo overhead, bisognerebbe preferire uno schema statico quando una proprietà fondamentale può spaziare in un insieme limitato di possibili valori discreti. Come mostrato in [**Eug01**], è possibile incrementare l'espressività introducendo il filtraggio basato sul contenuto in un contesto dove gli argomenti sono configurati staticamente, riuscendo così ad esprimere vincoli su proprietà i quali valori non appartengono ad un range discreto, come i prezzi delle azioni.

Capitolo 3

Il middleware SALES

Il progetto **S**calable context-**A**ware middleware for mobi**L**e **E**nvironment**S** [**Cor09**] (SALES), sviluppato dal gruppo di ricerca del DEIS, nasce con l'obiettivo di progettare ed implementare un middleware, ovvero un'infrastruttura di supporto, per sistemi mobili context-aware in grado di semplificare lo sviluppo delle applicazioni fornendo supporto a numerosi problemi: rappresentazione, accesso e distribuzione dei dati di contesto, eterogeneità dei dispositivi e delle comunicazioni wireless adottate, gestione della mobilità e della localizzazione, scalabilità, sicurezza e privacy.

Lo scopo di questo capitolo è quello di descrivere il middleware SALES, punto di partenza per la progettazione, l'implementazione e la fase di test di questa tesi. In particolare, il Paragrafo 3.1 espone i suddetti problemi supportati e delinea le caratteristiche principali del middleware. Nel Paragrafo 3.2 si descrive il principale scenario applicativo. Il Paragrafo 3.3 e il Paragrafo 3.4 definiscono, rispettivamente, l'architettura distribuita e l'architettura software di SALES. Nel Paragrafo 3.5 si descrive la distribuzione dei dati di contesto. Il Paragrafo 3.6 esprime la rappresentazione delle query di contesto. Infine, nel Paragrafo 3.7 si riporta la tecnologia utilizzata per implementare il middleware.

3.1 Caratteristiche principali

Il supporto fornito dal middleware SALES permette alle applicazioni di concentrarsi sulla logica applicativa senza preoccuparsi dei principali problemi dei sistemi context-aware. In particolare, SALES fornisce supporto per:

1. *Rappresentazione, accesso e distribuzione del contesto*: tutti i dati di contesto devono essere opportunamente rappresentati e distribuiti all'interno del sistema. D'altronde, ogni singolo dispositivo deve aver accesso continuo a tali dati per garantire la correttezza dell'adattamento sia del middleware che delle applicazioni. Ad esempio, considerando il servizio di stampa, il dispositivo deve poter aver accesso al descrittore associato al luogo in cui si trova e tale accesso deve essere garantito durante gli spostamenti. In particolare, la disseminazione e il reperimento dei dati di contesto è un problema fondamentale all'interno di tali sistemi dato che introduce una mole di traffico non banale e strettamente proporzionale alla popolazione corrente.

2. *Eterogeneità dei dispositivi e delle comunicazioni wireless adottate*: per supportare scenari pervasivi, il middleware deve supportare l'eterogeneità sia in termini di dispositivi sia di comunicazioni wireless. Ad esempio, all'interno di un'area universitaria possiamo trovare dispositivi molto differenti in termini di risorse, come tradizionali computer portatili e telefoni cellulari, che comunicano tra loro sfruttando standard wireless differenti, come IEEE 802.11 a/b/g (Wi-Fi) e Bluetooth (BT). Tutte queste differenti offerte devono essere considerate dal middleware per consentire adattamenti automatici e dinamici dipendenti dalla comunicazione wireless e dal dispositivo su cui è posto in esecuzione. Ad esempio, se il middleware deve fornire un servizio per un telefono cellulare con una semplice connessione BT, dovrà adattare automaticamente sia l'erogazione di servizi che i protocolli di gestione per evitare di saturare la banda.

3. *Gestione della mobilità e della localizzazione*: tutti i dispositivi coinvolti nello scenario principale sono mobili. Il middleware quindi ha il compito di gestire la mobilità per offrire continuità nell'erogazione dei servizi e per supportare la disseminazione dei dati verso i singoli dispositivi. D'altra parte, se la mobilità deve essere gestita per mantenere il collegamento logico al sistema, il middleware deve anche essere in grado di utilizzare un sistema di localizzazione dato che informazioni accurate su tale attributo possono essere sia utilizzate all'interno del contesto sia utilizzate per introdurre protocolli di gestione adattativi e basati sulla staticità del nodo.

4. *Scalabilità*: come accennato sopra, un middleware context-aware deve consentire la distribuzione efficace dei dati di contesto. Per garantire la scalabilità finale, il middleware dovrebbe sfruttare standard wireless differenti (in modo da utilizzare la banda presente su ognuno di essi) e dovrebbe anche sfruttare differenti modalità di comunicazione, sia basate su infrastrutture sia su comunicazioni ad-hoc. D'altronde, se le infrastrutture fisse permettono di garantire la distribuzione dei dati di contesto, le comunicazioni ad-hoc consentono lo scambio diretto di messaggi tra nodi senza introdurre traffico addizionale sull'infrastruttura. Inoltre, i dati dovrebbero essere distribuiti solo ai nodi che ne fanno richiesta evitando disseminazioni in aree non richieste. Infine, il middleware deve rafforzare i principi di località fisica e logica per ridurre sprechi di banda inutili. Ad esempio, gli avvisi emessi dal servizio di stampa dell'esempio riportato precedentemente dovrebbero essere mantenuti vicini al luogo in cui si trova la stampante dato che la loro utilità decresce in maniera forse più che proporzionale con la distanza (difficilmente l'utente è disposto a percorrere grandi distanze per reperire dei documenti stampati).

5. *Sicurezza e privacy*: le informazioni di contesto possono contenere dati sensibili da proteggere adeguatamente. Se necessario, il middleware deve garantire integrità, riservatezza e disponibilità dei dati, consen-

tendo l'accesso alle sole persone autorizzate. Per esempio, se il contesto contiene un attributo per la localizzazione attuale dell'utente, bisogna assolutamente garantire che tali dati siano disponibili solamente a livello di infrastruttura e magari soltanto ad alcuni utenti autorizzati, ad esempio agli amici, per evitare violazione della privacy. Per di più, la gestione della sicurezza non è per nulla semplice dato che, ad esempio, la protezione della riservatezza dei dati mediante crittografia potrebbe compromettere l'utilizzabilità dei meccanismi di distribuzione dei dati di contesto.

Tali oneri legati alla distribuzione dei dati di contesto vengono ridotti sfruttando alcune soluzioni innovative:

- *Architettura gerarchica ad albero*: sfruttando il principio di località fisica, SALES raggruppa i nodi vicini per ottenere un'architettura gerarchica distribuita ad albero, così da adattare la distribuzione dei dati di contesto. Inoltre, sfruttando il principio di località logica, SALES distribuisce i dati di contesto solo ai nodi interessati, riducendo così l'overhead introdotto e, contemporaneamente, assicurando l'accesso ai dati.

- *Comunicazioni wireless eterogenee*: SALES usa le diverse opportunità di comunicazione fornite da ogni singolo dispositivo per distribuire e reperire i dati di contesto. In particolare, SALES utilizza attivamente diversi standard wireless, come l'IEEE 802.11 (Wi-Fi) e il Bluetooth (BT), allo scopo di aumentare la banda disponibile e di massimizzare la copertura del sistema.

- *Modalità wireless eterogenee*: molte tecnologie wireless, come Wi-Fi e BT, consentono la comunicazione ad-hoc tra i dispositivi vicini. L'utilizzo di tale modalità consente a SALES di realizzare una distribuzione dei dati di contesto meno costosa, senza introdurre alcun overhead sull'infrastruttura wireless fissa.

- *Caching distribuito dei dati di contesto*: l'enorme quantità di dati di contesto può superare la capacità dei sistemi di memorizzazione di qualsiasi dimensione. Inoltre, il numero di richieste indirizzate per il reperimento dei dati di contesto può essere eccessivo per qualsiasi infrastruttura fissa/server. Per ridurre la memoria e l'overhead di elaborazione, la distribuzione dei dati, oltre alle tradizionali architetture basate su cluster utilizzate dall'infrastruttura fissa, usa i dispositivi mobili per costruire un repository distribuito di dati di contesto.

- *Adattività*: SALES utilizza soluzioni adattative per aumentare la scalabilità del sistema. Esso adatta i protocolli di gestione della mobilità in base al range delle trasmissioni wireless e ai pattern di mobilità dei dispositivi. Inoltre, adatta i protocolli di distribuzione dei dati in base alla banda corrente disponibile.

- *Distribuzione dei dati basata sulla QoC*: SALES sfrutta la QoC per imporre requisiti di qualità sia sui dati scambiati che sui processi stessi di distribuzione. SALES introduce i contratti, chiamati *Data Distribution Level Agreement* (CDDLA), per specificare i requisiti di qualità che bisogna garantire. Un'importante conseguenza è che, quando tali vincoli sono rispettati, SALES può adattarsi per massimizzare la scalabilità del sistema.

- *Distribuzione dei dati context-aware*: SALES sfrutta la consapevolezza del contesto nelle reti MANET per adattare il processo di distribuzione dei dati. In particolare, monitora la banda disponibile e stima lo stato dei canali wireless, così da prevenire congestioni e collisioni.

3.2 Scenario

Il principale scenario di SALES contiene diversi utenti che si muovono in maniera random con i loro dispositivi mobili, come PDA, laptop e telefoni cellulari. Ogni dispositivo esegue le proprie applicazioni context-aware

e usa un propria versione di SALES molto leggera per partecipare all'architettura distribuita finale. Ogni applicazione registra le proprie esigenze di contesto all'istanza locale di SALES che, a sua volta, si occuperà del reperimento e della distribuzione dei dati di contesto. Inoltre, ogni applicazione può registrare opportune sorgenti dati con il principale obiettivo di generare e disseminare nuovi dati di contesto. Come i tradizionali sistemi mobili, SALES adotta un'approccio basato su un'infrastruttura per realizzare la distribuzione del contesto e per garantire l'accesso a tali dati. Come aspetto innovativo, SALES sfrutta anche le comunicazioni ad-hoc per ridurre il peso della distribuzione dei dati di contesto. L'utilizzo congiunto di tali opportunità aumenta la scalabilità, in quanto il sistema avrà una banda finale più ampia, e la copertura di sistema, poiché viene assicurato l'accesso ai dispositivi mobili indipendentemente dalle capacità specifiche del dispositivo (ad esempio, un tradizionale telefono cellulare con la sola interfaccia Bluetooth può ottenere i servizi offerti dall'infrastruttura SALES sfruttando come router intermediario un laptop con due interfacce). La Figura 3.1 mostra quanto detto a parole.

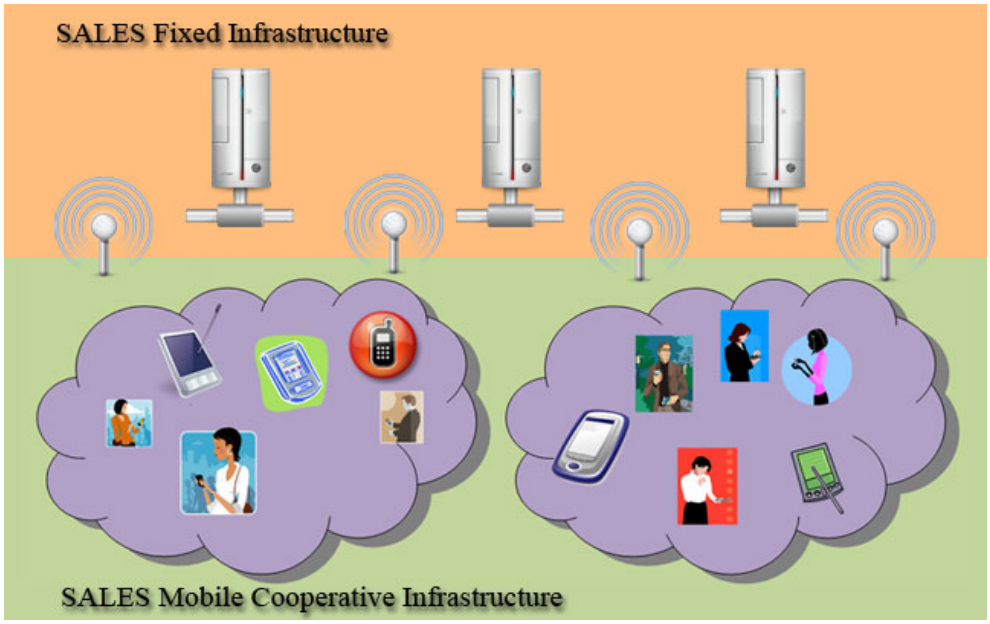

Figura 3.1: Scenario di cooperazione tra l'infrastruttura fissa e le comunicazione ad-hoc.

Si precisa che la presenza dell'infrastruttura non è obbligatoria per il corretto funzionamento, ma rappresenta l'unica scelta per garantire l'accesso ai dati di contesto. Per realizzare la distribuzione dei dati di contesto tramite i collegamenti ad-hoc, SALES sfrutta il principio di località fisica per raggruppare i nodi utente vicini. Ogni gruppo rappresenta un ambiente di cooperazione all'interno del quale ogni dispositivo mobile condivide il proprio repository di dati di contesto fungendo da sorgente dati per gli altri nodi. Come spiegherà meglio il Paragrafo 3.3, l'astrazione di gruppo è utile per adattare la distribuzione dei dati di contesto e per ridurre la gestione dello scope delle informazioni. In conclusione, il principale scenario può essere rappresentato in questo modo: una o più infrastrutture wireless fisse abilitano l'accesso ai dati di contesto da parte dei dispositivi mobili. Contemporaneamente, tali dispositivi mobili si adattano tra loro per costruire un gruppo di cooperazione nel quale i dati di contesto sono scambiati utilizzando comunicazioni ad-hoc. Infine, nodi dotati di doppia interfaccia fungono da router per collegare assieme le differenti reti wireless coinvolte.

3.3 Architettura distribuita

L'architettura distribuita di SALES può essere suddivisa in due macrolivelli principali. Il livello superiore si basa su nodi fissi e potenti utili per ottenere una distribuzione veloce e affidabile dei dati di contesto, mentre il livello inferiore sfrutta dispositivi meno potenti, che comunicano mediante collegamenti wireless ad-hoc, per ottenere una maggiore affidabilità ed effettuare una distribuzione meno costosa. Infine, dispositivi utente 'versatili' dotati di doppia interfaccia collegano i due livelli per realizzare un sistema unico in cui disseminare e reperire i dati di contesto.

Come mostra la Figura 3.2, il middleware distingue quattro tipi di nodi differenti:

1. **Central Node (CN)**: è la radice dell'architettura distribuita. Essa è l'unica entità statica delegata alla memorizzazione persistente dei

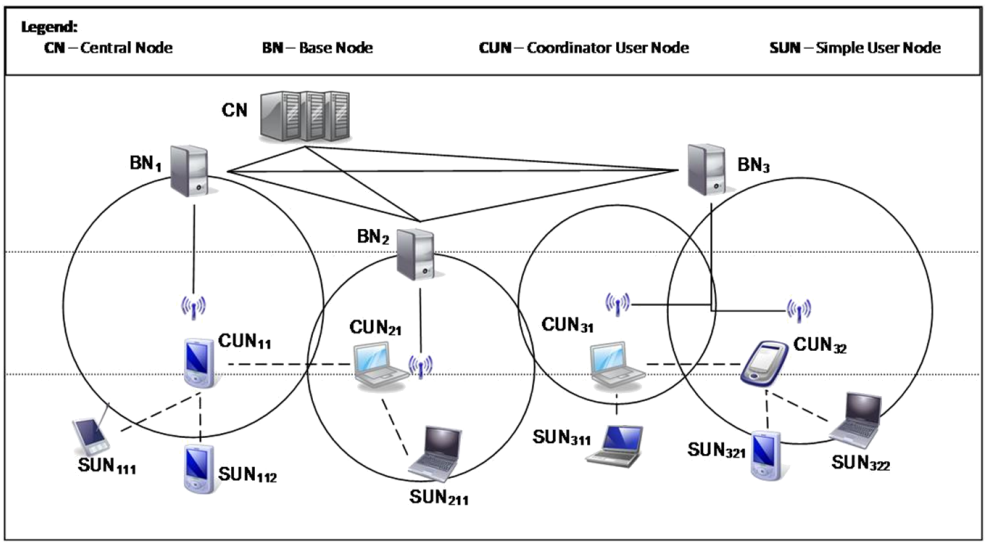

Figura 3.2: Architettura distribuita di SALES.

dati di contesto e fornisce i dati solo su richiesta; in quanto tale, è tipicamente realizzata in architettura cluster per garantire la scalabilità. Il CN è nascosto dai nodi appartenenti al livello inferiore (BN) ed è quindi in grado di scambiare messaggi solo con essi.

2. **Base node (BN)**: ognuno di essi è un nodo fisso associato a uno o più Access Point Wi-Fi disponibili nell'ambiente. Un BN garantisce la connessione con i nodi utenti presenti nei luoghi serviti e si comporta da cache dei dati di contesto per ridurre le richieste verso il CN. Ogni BN condivide inoltre tali cache locali con i propri pari per ridurre ulteriormente le richieste inviate al livello superiore. In base agli specifici dati di contesto, i BN possono adottare politiche sia reattive che proattive per la gestione dei dati dal CN.

3. **Coordinator User Node (CUN)**: i dispositivi utenti sono riuniti in gruppi per favorire la disseminazione dei dati di contesto. Ogni gruppo prevede un coordinatore locale eletto dinamicamente, ossia un CUN, che controlla la disseminazione dei dati e la mobilità all'interno del gruppo. Essendo anche il coordinatore un dispositivo mobile, ogni CUN esegue protocolli di gestione della mobilità con il suo BN, se disponibile, e con i nodi serviti per permettere il discovery

e la formazione dei gruppi. SALES sfrutta inoltre i CUN per costruire un'infrastruttura di instradamento multi-hop e basata sul contesto, utile per ridurre sia il carico introdotto nei livelli superiori sia i tempi di risposta. Questa infrastruttura di instradamento è inoltre l'unica scelta disponibile per sostenere la disseminazione e la ricerca dei dati in scenari privi di infrastruttura. Come detto prima, i CUN sono tradizionalmente dispositivi dotati di doppia interfaccia dato che gestiscono contemporaneamente sia le comunicazioni basate su infrastruttura che quelle con modalità ad-hoc.

4. **Simple User Node (SUN)**: ogni dispositivi utente che non è un coordinatore è un SUN. Ogni SUN, come un tradizionale CUN, interagisce producendo e recuperando dati di contesto. Inoltre, un SUN può scambiare messaggi solo con il proprio CUN e con i propri pari mediante comunicazioni ad-hoc, e deve mantenere un'associazione con un CUN disponibile per avere accesso al sistema.

3.4 Architettura software

Il middleware deve risolvere differenti problemi legati allo scenario principale supportato. In prima istanza, SALES deve gestire la mobilità, l'eterogeneità delle comunicazioni wireless e la scalabilità del sistema finale. Partendo dalla gestione della mobilità, sia i SUN che i CUN devono effettuare la procedura di discovery dell'infrastruttura presente e degli altri nodi disponibili. Attraverso tale procedura si identificano i nodi a visibilità diretta. Inoltre, ogni singolo nodo utente deve monitorare continuamente il collegamento logico con il nodo padre, introducendo un overhead ulteriore in termini di banda. Pertanto, SALES cerca di adattare i protocolli di gestione della mobilità in base alla copertura garantita dallo standard wireless utilizzato e alla mobilità dei singoli dispositivi, allo scopo di aumentare il periodo di verifica del collegamento logico con il padre e ridurre quindi il consumo di banda. Inoltre, SALES deve evitare la saturazione dei canali wireless, adattando la trasmissione dei dati in base alla banda

attuale, e deve sfruttare le differenti modalità, ad-hoc e basata su infrastruttura, per ridurre il carico introdotto e aumentare la scalabilità. Infine, dato che la disseminazione dei dati di contesto introduce un overhead direttamente proporzionale alla popolazione corrente ed ai dati distribuiti, SALES garantisce la scalabilità disseminando solo i dati necessari ed evitando politiche di disseminazione basate su inondazione. In particolare, il middleware realizza percorsi di instradamento dei dati mediante il concetto di 'context query'. Le context query rappresentano le esigenze del livello applicativo e devono essere propagate per la disseminazione dei dati all'interno dell'architettura distribuita di SALES. Attualmente, le query possono adottare anche tecniche probabilistiche, quali i filtri di Bloom (spiegati nel Paragrafo 3.6), per ridurre la dimensione finale dei messaggi utilizzati per la disseminazione.

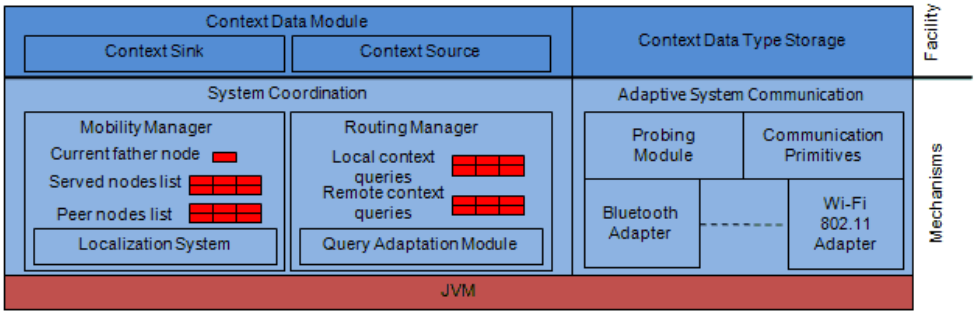

Figura 3.3: Architettura software di SALES.

Da un punto di vista di alto livello, SALES adotta l'architettura software a due livelli mostrata in Figura 3.3:

- **Livello Facility**: questo livello implementa tutte quelle funzionalità di alto livello necessarie alla creazione, al reperimento e alla distribuzione dei dati di contesto. Ogni dato di contesto è associato univocamente ad un apposito descrittore che fornisce sia la struttura del dato che le politiche di disseminazione per quel particolare tipo di dato. Il *Context Data Type Storage* consente la memorizzazione e l'accesso a tali descrittori. Ogni dato di contesto è associato ad un appropriato *Context Data Module* che contiene una sorgente ed una sink: la

Context Source consente la creazione e la distribuzione di nuovi dati di contesto, mentre la *Context Sink* permette il reperimento dei dati di contesto mediante le context query.

- **Livello Mechanisms**: questo livello realizza tutte le funzionalità di basso livello utili alla comunicazione inter-nodo e alla formazione della gerarchia. L'*Adaptive System Communication* offre primitive generiche per l'invio e la ricezione dei messaggi, nasconde i dettagli delle singola tecnologie wireless e impone il rispetto della banda disponibile. Inoltre, ogni tecnologia wireless ha un apposito adattatore, come il *Bluetooth Adapter* e il *Wi-Fi Adapter*, che implementa le reali funzionalità di invio/ricezione sulla specifica tecnologia. Sopra tali adapter, il *Communication Primitives* realizza primitive di invio sia one-way che request/response, mentre il *Probing Module* monitora periodicamente la banda disponibile su ogni collegamento. Il *System Coordination* utilizza le *Application Programming Interface* (API) di comunicazione fornite dall'*Adaptive System Communication* per supportare la mobilità e le decisioni di instradamento. Il *Mobility Manager* sfrutta il *Localization System* per ritrovare la localizzazione attuale e per realizzare protocolli di mobilità adattativi. Infine, il *Routing Manager* impone le strategie di distribuzione dei dati e delle query e usa il *Query Adaptation Module* per adattare dinamicamente la rappresentazione delle query.

3.5 Distribuzione dei dati di contesto

In SALES ogni applicazione definisce i propri tipi di dato di contesto; ogni definizione deve contenere la struttura di un'istanza generica di dato di contesto e specificare le politiche di distribuzione per limitare la visibilità dei dati all'interno dell'architettura. In particolare, il tipo di dato utilizzato per le simulazioni effettuate per questa tesi ha la struttura presentata in Figura 3.4, dove sono presenti 2 coppie attributo-valore ed un *payload*, ovvero l'informazione vera e propria.

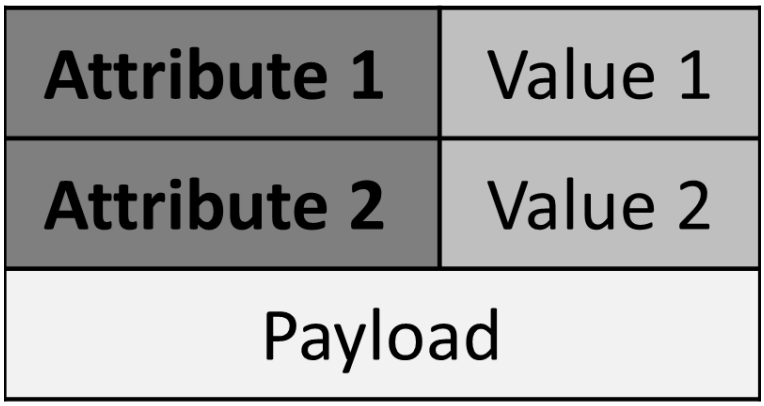

Figura 3.4: Struttura dei dati utilizzati in questa tesi.

Dopo aver definito i dati, i nodi mobili possono iniettare nuove istanze di dato nell'architettura e possono emettere le richieste di contesto in accordo con le proprie esigenze. In questa tesi le richieste sono rappresentate mediante combinazioni logiche di coppie attributo-valori consentendo di discriminare i dati a partire da un insieme di valori possibili per uno o più attributi.

SALES discrimina tra *context data instances* e *context queries*. Mentre il primo rappresenta il reale dato di contesto, il secondo rappresenta le informazioni di instradamento utili per guidare la distribuzione delle istanze del dato. La traduzione degli interessi dell'applicazione in query di contesto è effettuata dalla Context Sink contenuta in ogni Context Data Module.

Di base, sia le istanze di dato che le query di contesto sono disseminate all'interno del sistema seguendo un cammino dal basso verso l'alto, dal nodo creatore al CN. Tali cammini sono chiamati rispettivamente *data dissemination path* e *query dissemination path*. Inoltre, per incrementare la probabilità di trovare i dati cercati nei livelli più bassi dell'architettura, le query di contesto sono disseminate anche orizzontalmente prima che verso l'alto. Quando si riceve una nuova context query, il *Routing Manager* scatena la verifica della corrispondenza data/query con i dati memorizzati localmente ed inoltra i dati che soddisfano la query verso il nodo creatore

della stessa. L'instradamento dei dati di contesto è realizzato hop-by-hop e ogni nodo può inviare i dati solo ai pari a visibilità diretta (distanza one-hop, come il nodo padre, nodi figli e nodi pari). Quando un nodo riceve una nuova istanza di dato di contesto, verifica la corrispondenza con le query di contesto memorizzate ed, eventualmente, scatena la fase di disseminazione. Tale fase prevede l'invio della risposta al nodo dal quale si è ricevuta la query senza alcuna informazione aggiuntiva che specifichi la query di riferimento. Quindi, il nodo che riceverà il dato di risposta effettuerà le stesse operazioni del nodo precedente a partire dal confronto tra il dato ricevuto e le query di contesto memorizzate.

Sia le istanze di dato di contesto che le query di contesto sono associate con la gestione dei dati, utile per modificare il cammino finale di disseminazione. In particolare, ogni istanza di dato di contesto è associata al seguente insieme di parametri:

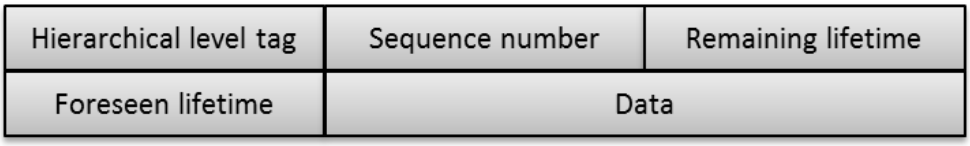

Figura 3.5: Rappresentazione dei dati di contesto.

- **Hierarchy Level Tag**: questo parametro limita la distribuzione verticale del dato all'interno dell'architettura ad albero SALES con una granularità d'istanza. Esso può limitare la disseminazione fino al livello CUN, BN oppure CN.
- **Sequence number**: numero positivo crescente associato dalla sorgente e usato per discriminare i vecchi dati da quelli nuovi.
- **Remaining lifetime**: numero positivo dinamicamente decrementato e usato per capire se un'istanza di dato sia ancora valida o meno.
- **Foreseen lifetime**: tempo di vita iniziale dell'istanza di dato associato dalla sorgente.

Figura 3.6: Rappresentazione delle query di contesto.

Mentre ogni query di contesto è associata ai seguenti:

- **Query lifetime**: deadline assoluta dopo la quale la query scade e perciò viene eliminata dal supporto.
- **Query Routing Delay**: massimo ritardo applicato ad ogni nodo durante l'instradamento della query.
- **Data Routing Delay**: massimo ritardo applicato da ogni nodo durante l'instradamento del dato in risposta alla query.
- **Query Priority**: priorità della query utilizzata per differenziare il traffico.
- **Horizontal Time To Live**: il numero massimo di nodi attraversati allo stesso livello gerarchico.
- **Maximum Query Responses**: il numero massimo di risposte desiderate.
- **Query Level Mask**: maschera binaria utilizzata per esprimere una limitazione sui livelli SALES che possono o non possono fornire i dati di contesto in risposta alla query.
- **Data Filters**: i filtri utilizzati per selezionare i dati di contesto.

3.5.1 Distribuzione basata su QoC

La QoC si applica tipicamente alla qualità dei dati di contesto distribuiti. Invece SALES usa questo concetto per adattare la distribuzione e la consegna dei dati di contesto, così da preservare le risorse del livello precedente. Per guidare le operazioni di gestione e di riconfigurazione, sono

state introdotti dei contratti speciali, ovvero i *Context Data Distribution Level Agreements* (CDDLA) come anticipato nel Paragrafo 3.1, per descrivere il livello di qualità che l'infrastruttura deve garantire.

Ogni CDDLA vincola tre principali parametri:

1. **Freshness**: indica la freschezza dei dati e permette di richiedere solo i dati più aggiornati oppure anche quelli meno recenti.
2. **Data retrieval time**: indica il tempo massimo che il nodo creatore della query è disposto ad aspettare per ricevere il dato di contesto.
3. **Priority**: ha il tradizionale significato di priorità, utile per differenziare il traffico e per favorire l'instradamento di dati più prioritari in caso di sovraccarico.

Per chiarezza, si introducono le tre principali classi utente che definiscono i parametri di qualità ad essi associati:

1. **Gold**: [ultima versione del dato, 2 secondi, alta]
2. **Silver**: [dato valido, 4 secondi, media]
3. **Bronze**: [dato scaduto da al massimo 2 secondi, 6 secondi, bassa]

3.6 Rappresentazione delle query

Per adattare la distribuzione delle query in accordo con il carico di rete corrente, l'*Adaptive System Communication* monitora lo stato dei canali di ogni collegamento wireless e impone delle limitazioni di banda per le operazioni d'invio. Attraverso il monitoraggio dei messaggi pendenti e della banda, SALES ottiene indicazioni sullo stato corrente del canale che possono essere utili per adattare la distribuzione dei dati di contesto e per scatenare l'adattamento delle query.

In particolare, SALES riduce i requisiti di banda adattando la rappresentazione delle query. Dato un particolare tipo di dato, una query di

contesto rappresenta delle esigenze verso i dati imponendo dei vincoli sui valori assunti dagli attributi delle istanze di dato. Tradizionalmente ogni query di contesto contiene un set di vincoli di base, per esempio *membership test* (test di appartenenza), test di range e altri, combinati con gli operatori logici AND/OR; la query viene trasmessa mediante meccanismi tradizionali di serializzazione. Quando il canale wireless è molto congestionato e la query contiene dei membership test, è possibile sfruttare dei particolari filtri chiamati filtri di Bloom.

Un filtro di Bloom è una struttura dati probabilistica efficiente dal punto di vista spaziale, utile per supportare membership query su un set $A = \{a_1, a_2, , a_n\}$ di n chiavi. Ogni filtro è un vettore di m bit, inizialmente tutti settato a 0, calcolati applicando k funzioni hash indipendenti $\{h_1, h_2, ., h_k\}$ ad ogni chiave nell'insieme originale. Il filtro associato all'insieme di chiavi è ottenuto impostando a 1 tutti i bit in posizione $h_1(a), h_2(a), ..., h_k(a)$ per ogni elemento a di A. Data una query per una generica chiave b, si verificano tutti i bit di posizione $h_1(b), h_2(b), ..., h_k(b)$ e, se almeno uno di essi è uguale a 0, allora b non sarà sicuramente contenuto nell'insieme originale. Altrimenti, si assume che b è contenuto nell'insieme, sebbene potrebbero verificarsi falsi positivi. Se sono coinvolti membership test e se è possibile definire delle funzioni hash per gli attributi coinvolti, il *Query Adaptation Module* può trasformare i test introducendo i filtri di Bloom.

In particolare, come mostra la Figura 3.7 utilizzando i simboli convenzionali || per l'OR e $\&\&$ per l'AND, SALES adotta i filtri di Bloom in due modi differenti:

1. **Single Constraint - Single Filter (SCSF)**: questa modalità mappa ogni membership test in una coppia nome-filtro appropriata.

2. **Multiple Constraints - Single Filter (MCSF)**: questa modalità mappa differenti membership test (legati dallo stesso operatore logico) in soli due filtri di Bloom. Il primo filtro è usato per selezionare gli at-

tributi coinvolti, mentre il secondo è utilizzato per vincolare i valori degli attributi.

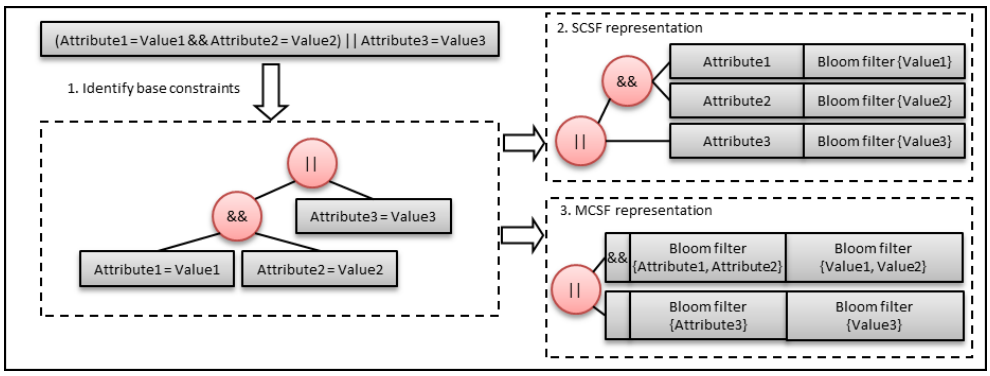

Figura 3.7: Rappresentazione delle query di contesto.

Le due modalità sono usate insieme a quelle priva di compressione e sono adottate dinamicamente durante l'instradamento del messaggio.

3.7 Tecnologia d'implementazione

Il middleware SALES è stato inizialmente implementato in linguaggio $J2SE$ 1.6 ed in seguito è stata realizzata una versione adattata per la piattaforma mobile Android.

Capitolo 4

Sintetizzazione e aggregazione delle query

I servizi di diffusione delle informazioni basati sul modello publish/subscribe risultano essere adatti alle necessità degli utenti, ma possono essere al contempo molto costosi [**Cre03**].

Questo capitolo indaga sulle principali tecniche di riduzione della banda e della complessità di comunicazione nei sistemi pub/sub, per poter estrarre, adattare ed applicare tali idee e concetti ai sistemi context-aware ed in particolare all'infrastruttura SALES, al fine di ridurre il carico di lavoro dei singoli nodi ed il traffico dei dati di contesto generato in risposta alle richieste. Le due principali tecniche presentate sono: sintetizzazione delle query [**Tri04**], analizzata nel Paragrafo 4.1, e aggregazione delle query [**Cre03**], analizzata nel Paragrafo 4.2.

4.1 Sintetizzazione delle query

Si considerino i sistemi pub/sub nei quali gli eventi pubblicati da un publisher, connesso ad un qualsiasi broker, potrebbero essere d'interesse per qualunque subscriber, connesso ad un qualsiasi broker. La Figura 4.1 mostra un'architettura pub/sub distribuita.
La progettazione e l'implementazione di sistemi pub/sub su larga scala

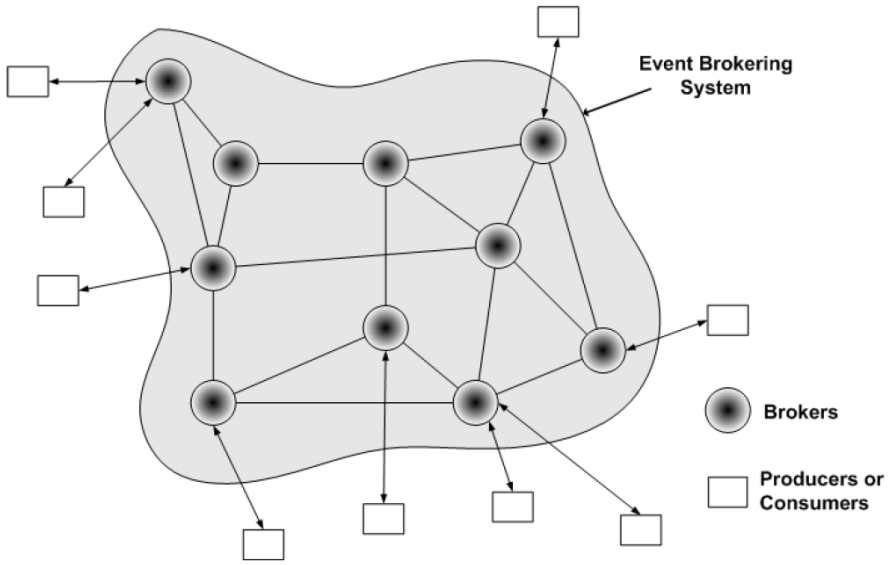

Figura 4.1: Architettura distribuita di un sistema pub/sub.

dovrebbero garantire l'efficienza durante le due fasi concorrenti di esecuzione. La prima fase riguarda la propagazione degli interessi verso un numero potenzialmente elevato di broker. Questi ultimi necessitano di tali informazioni per inoltrare gli eventi in ingresso solo agli utenti interessati e per filtrare gli eventi non rilevanti, preservando così sia la banda che il tempo di elaborazione. La seconda fase si riferisce all'elaborazione di un nuovo evento, la quale ha il compito di filtrarlo oppure, in caso di corrispondenza, di inoltrarlo ai broker i quali hanno precedentemente espresso interesse per esso.

In [**Tri04**] viene introdotto il concetto di sintetizzazione degli interessi, il quale prevede la compattazione di tutti gli interessi ricevuti da un broker. In particolare, ogni interesse è formato da un insieme di coppie chiave-valore le quali vengono suddivise e inserite ognuna all'interno di un'apposita struttura di sintetizzazione. Ad esempio, nel caso in cui si ricevessero le due espressioni di interesse mostrate in Figura 4.2, bisognerebbe considerare separatamente ogni sotto-interesse e sintetizzare ognuno di essi in un'opportuna struttura.

Interest 1			**Interest 2**		
Type	*Name*	*Constraint*	*Type*	*Name*	*Constraint*
string	exchange	N*SE	string	symbol	>* OT
string	symbol	= OTE	float	price	= 8.20
float	price	< 8.70	integer	volume	> 130000
float	price	> 8.30	float	low	< 8.05

Figura 4.2: Esempio di interessi ricevuti da un broker.

La figura 4.3 riporta l'esempio del prezzo.

Summary for attribute *price*			
Range	*min*	*max*	*id list*
	8.30	8.70	→ **I1**
Equality	*value*		*id list*
	8.20		→ **I2**

Figura 4.3: Esempio di struttura di sintetizzazione del sotto-interesse *prezzo*.

Tali strutture di sintetizzazione devono essere condivise tra i broker al fine di distribuire tutti gli interessi. Per fare ciò si utilizza l'algoritmo presentato in Figura 4.4, dove per *degree* s'intende il numero di broker con i quali un nodo è collegato direttamente.

For *i*=1 to MAX_DEGREE
Each node of degree *i*:
1. Merges its summary with all received summaries and updates the set *Merged_Brokers*.
2. Sends the merged summary and the set *Merged_Brokers* to a selected neighbor with equal or higher degree.

End

Figura 4.4: Algoritmo di propagazione delle strutture di sintetizzazione.

Tale algoritmo viene eseguito su ogni nodo periodicamente e al suo termine tutti i broker hanno le stesse identiche strutture di sintetizzazione.

Dopo aver terminato la prima fase, i broker sono pronti a ricevere gli eventi e all'atto della ricezione bisogna effettuare le seguenti operazioni:

1. Identificare tutti gli attributi dell'evento.

2. Per ogni attributo, verificare se all'interno della struttura di sintetizzazione degli interessi vi sia almeno un interesse soddisfatto.

3. Se per tutti gli attributi dell'evento esiste almeno un interesse sempre soddisfatto, allora l'evento viene inviato al mittente di tale interesse, broker o subscriber che sia.

La tecnica di sintetizzazione, come afferma l'articolo [**Tri04**], assicura dei benefici sia in termini di banda richiesta per propagare gli interessi agli altri broker e sia in termini di overhead di memorizzazione su ogni broker.

La sintetizzazione delle query, seppur molto efficace per il modello publish/subscribe, mal si adatta ad architetture context-aware poiché in queste ultime vengono coinvolti nella distribuzione dei dati di contesto dispositivi con qualunque capacità computazionale. Questo implica che la nostra soluzione deve essere in grado di fornire comportamenti differenziati tra i vari dispositivi che prendono parte alla comunicazione, sia in base al ruolo che giocano nell'infrastruttura e sia in base alle capacità computazionali. Tale soluzione, al contrario, impone una distribuzione delle strutture di sintetizzazione tra tutti i dispositivi coinvolti, obbligando di fatto tutti i nodi ad avere lo stesso comportamento. Inoltre per i sistemi context-aware non è vero che tutti i nodi ricevono le stesse query; in SALES, ad esempio, due nodi coordinatori (CUN) non riceveranno mai una stessa query, a meno che non siano in visibilità orizzontale.

4.2 Aggregazione delle query

In [**Cre03**] si afferma che il consumo eccessivo di risorse nei servizi di disseminazione basati sugli interessi è il risultato di tre fattori:

1. La rete è point-to-point, ovvero le risposte ad ogni query vengono trasmesse separatamente per ogni subscriber.

2. Ogni query viene processata individualmente.

3. I subscriber vengono considerati dei processi 'dumb' che non sono in grado di effettuare nessuna operazione di filtraggio sugli eventi che ricevono.

L'overhead della disseminazione point-to-point può essere ridotto mediante l'utilizzo di una rete multicast. Considerando ad esempio il caso fortunato in cui n subscriber emettono esattamente la stessa query per esprimere i loro interessi, con una rete point-to-point la query sarà elaborata n volte e le risposte saranno inviate n volte, mentre con una rete multicast si stabilisce una 'canale' per la risposta consentendo di inviarla una volta sola. In molte applicazioni però non è comune che un elevato numero di utenti emettano la stessa query, quindi il solo utilizzo di una rete multicast non sempre consente di ottenere un grande vantaggio.

Se invece si considera anche l'aggregazione di query i quali insiemi di risposte si sovrappongono significativamente, l'utilizzo della rete multicast risulta più facilmente applicabile e quindi il vantaggio è maggiore. Mediante l'aggregazione di tali query, il server deve processare un numero inferiore di query e l'insieme delle informazioni inviate in risposta potrebbe essere ridotto. Un punto debole di tale approccio è rappresentato dal fatto che l'insieme delle risposte alla query aggregata potrebbe contenere alcuni dati non rilevanti per i subscriber, frutto del rilassamento dei vincoli della query. Per tale motivo il subscriber deve effettuare un filtraggio sui dati utilizzando la query originale, comportando un utilizzo di potenza

computazionale.

A titolo di esempio si consideri le query $q_1 : \sigma_{2 \leq A \leq 40} R(A)$ e $q_2 : \sigma_{3 \leq A \leq 41} R(A)$ e la loro aggregazione in $q_3 : \sigma_{2 \leq A \leq 41} R(A)$. Il server processa un'unica query ed invia il risultato, $risp(q_3)$, ai subscriber che hanno emesso le query originali. Il subscriber di q_1 deve estrarre la risposta da $risp(q_3)$ applicando la query di estrazione $q_1 : \sigma_{A \leq 40} R(A)$. Allo stesso modo, il subscriber di q_2 deve estrarre la query per eliminare tutti gli elementi inferiori a tre. L'aggregazione di q_1 e q_2 consente di ridurre sia il carico di lavoro fatto dal server per elaborare le query e spedire le risposte e sia il carico di informazioni inviate ai subscriber; il costo aggiunto riguarda la creazione di un canale multicast e la necessità dei subscriber di filtrare le risposte.

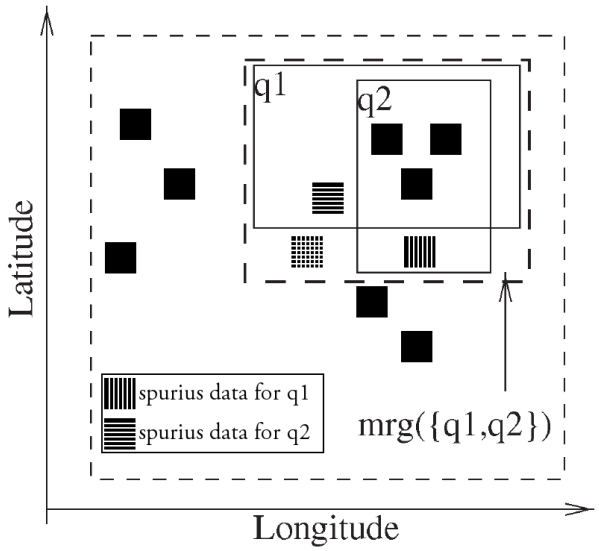

Figura 4.5: Esempio di aggregazione di due query.

La Figura 4.5 mostra quanto detto attraverso le query geografiche, scelte come esempio per la loro facilità di comprensione.

4.2.1 Modello concettuale

La Figura 4.6 mostra il modello concettuale di un servizio pub/sub.

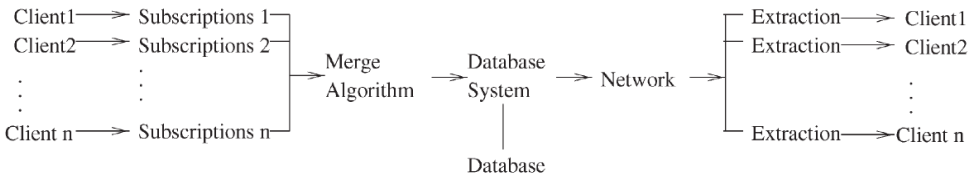

Figura 4.6: Modello concettuale di un servizio pub/sub.

In tale modello si ha un insieme di client $C = \{c_1, ..., c_n\}$ che richiedono informazioni, ognuno mediante un insieme di query. Quindi ogni client c_i genera l'insieme di query Q_i e l'insieme di tutte le query emesse da tutti i client viene chiamato Q. I client inviano i loro insiemi di query al server il quale li elabora periodicamente su un database ed invia le risposte ai mittenti. Prima di elaborare le query, il server esegue l'algoritmo di aggregazione che combina le query 'simili'. L'output del processo di elaborazione è una collezione $A = \{A_i\}$ dove ogni A_i contiene le query che sono state aggregate. Da ogni insieme A_i si genera un'unica query $agg(A_i)$, frutto dell'aggregazione di tutte le query originali contenute all'interno dell'insieme A_i. L'insieme delle risposte alla query q viene rappresentato come $risp(q)$, perciò il server genera $risp(agg(A_i))$ per ogni A_i contenuto in A. Per completezza si richiede che $\cup_i Q_i = \cup_i A_i$. In maniera simile, si richiede che $risp(q) \subseteq risp(agg(A_i))$ per ogni $q \in A_i$. La differenza tra la risposta alla query aggregata $risp(agg(A_i))$ e quella alla query originale $risp(q)$ viene definita come *informazioni irrilevanti per q* inviate al client.

Sebbene in SALES esista un server centrale, il nodo CN, che riceve tutte le query e tutti i dati generati dall'infrastruttura, si vuole sfruttare il fatto che ogni nodo, SUN a parte, sia in grado di rispondere alle richieste attraverso dati generati o ricevuti. Perciò questi nodi agiscono sia da client, quando inoltrano i dati e le query ai nodi del livello superiore, che da server, quando ricevono i dati e le query ed inoltrano le risposte ai nodi del livello inferiore. Ad esempio, i nodi CUN agiscono da client verso i nodi BN e da server verso i nodi SUN. Per questo motivo si ritiene che tutti i nodi CN, BN e CUN potrebbero trarre vantaggio aggregando le query ricevute. In particolare, i nodi CUN dovrebbero adattare l'aggregazione

per ridurre il proprio carico di lavoro in quanto nodi mobili e quindi dotati di scarse risorse computazionali. I nodi BN ed il nodo CN dovrebbero invece adattare l'aggregazione allo scopo di ridurre il traffico delle risposte verso i nodi mobili riducendo così anche il lavoro complessivo di questi ultimi. Di seguito saranno utilizzati i termini client e server per distinguere i ruoli che ogni nodo può interpretare e non per identificare particolari nodi.

Per illustrare i concetti inerenti all'aggregazione, si consideri l'emissione della query $Q_1 = \{x, y\}$ da parte del client c_1 e l'emissione della query $Q_2 = \{z\}$ dal client c_2. Il server riceve i due insieme e aggrega le query in $A_1 = \{x, z\}$ e $A_2 = \{y\}$, producendo due singole query $agg(A_1)$ e $agg(A_2)$. Queste ultime vengono processate su un database e generano in risposta $R_1 = risp(agg(A_1))$ e $R_2 = risp(agg(A_2))$. Si noti che R_1 deve essere inviato sia a c_1 che a c_2 ed entrambi devono applicare una *funzione di estrazione* per ottenere le sole risposte desiderate. Ad esempio, la funzione di estrazione e che c_1 applica ad R_1 dovrebbe generare $e(R_1) = risp(x)$. Per questo motivo, quando il server invia R_i deve includere un *header* contenente le seguenti informazioni:

1. Una lista di client che devono ricevere R_i.
2. Per ogni client c coinvolto, una o più coppie (e, q), dove e rappresenta la funzione di estrazione e q è l'identificatore della query. La funzione di estrazione e rappresenta quello che il client c deve applicare per ottenere la risposta alla sua query originali q.

Nell'esempio presentato le informazioni inviate con R_1 saranno $c_1 : (e_x, x)$ e $c_2 : (e_z, z)$. Il client c_1 deve applicare $e_x(R_1)$ per estrarre la risposta alla query x mentre il client c_2 deve applicare $e_z(R_1)$ per ottenere la sua risposta.

Ci sono molti modi per implementare la funzione di estrazione ma in alcuni casi basta utilizzare la query originale; in particolare, questo è vero quando le query considerano solo le selezioni e le proiezioni. In questo

caso non è necessario che il server invii la funzione di estrazione al client poiché quest'ultimo ne è già in possesso.

Si osservi che in SALES, come riportato nel Paragrafo 3.5, le risposte viaggiano senza alcuna informazione sulla query di riferimento; ogni nodo quando riceve un dato lo confronta con qualunque query memorizzata, svolgendo di fatto una funzione di estrazione. Per questo motivo si assume che l'estrazione dei dati utili da tutti quelli ricevuti sia un comportamento già implementato da tutti i nodi del middleware SALES.

4.2.2 Il modello di costo

Come accennato nel Paragrafo 4.2, le risposte alle query aggregate potrebbero contenere dei dati non rilevanti per i client, chiamati dati spuri. Questo potrebbe portare, in alcune situazioni, ad un peggioramento delle performance del sistema anziché un miglioramento. Per decidere se l'aggregazione apporta benefici o meno, si necessita di un preciso ed accurato modello di costo che consenta di confrontare i 'costi' del sistema quando sfrutta l'aggregazione con i costi del sistema senza aggregazione.

Il server riceve un insieme di query Q e produce in uscita un insieme A dove ogni elemento è un insieme di query da aggregare. Il problema dell'aggregazione delle query è quello di trovare l'insieme A di costo minimo. Gli input del problema sono una funzione di costo $cost()$, una procedura di aggregazione $agg()$ ed un insieme di query Q. L'output è una collezione A tale che il costo totale $cost(A)$ sia minimo.

Il costo di elaborazione delle query e di invio delle risposte è rappresentato dal totale delle risorse consumate. Tali risorse sono la somma di tutte quelle utilizzate dal server, dalla rete e dai client. I costi coinvolti nel modello possono essere così riassunti:

1. Costo del server per eseguire l'algoritmo di aggregazione e per elaborare le query aggregate.

2. Costo di trasmissione delle risposte alle query aggregate.

3. Costo dei client per eseguire la procedura di estrazione.

Per calcolare le risorse utilizzate bisogna stimare il costo di elaborazione delle query e la dimensione delle loro risposte. Si utilizza la notazione $cost(q)$ per rappresentare la stima di costo per recuperare le risposte alla query q. Il costo totale stimato per recuperare tutte le risposte sarà

$$cost(A) = cost(agg(A_1)) + cost(agg(A_2)) + ... + cost(agg(A_m)).$$

Per indicare la stima della dimensione della risposta a q si utilizza la notazione $size(q)$ e quindi la dimensione totale di tutte le risposte sarà

$$size(A) = size(agg(A_1)) + size(agg(A_2)) + ... + size(agg(A_m)).$$

La dimensione delle informazioni irrilevanti per la query q_i si denota con $u_i = size(agg(A_j)) - size(q_i)$ dato $q_i \in A_j$. La somma di tutti gli u_i sarà

$$U(Q, A) = \sum_{q_i \in Q} u_i.$$

Per modelli che prevedono la presenza di un server centralizzato molto potente, il costo di esecuzione dell'algoritmo di aggregazione può essere considerato insignificante. In questa tesi invece bisogna tenere conto di tale costo in quanto si vuole realizzare una soluzione eseguibile anche su dispositivi mobili. Si è ritenuto fondamentale considerare due fasi di elaborazione ben distinte: la *fase di aggregazione* e la *fase di interrogazione*. La prima riguarda l'esecuzione dell'algoritmo di aggregazione ed ha lo scopo di massimizzare il guadagno nella fase di interrogazione. Quest'ultima rappresenta la ricezione dei dati, la verifica di quali query siano soddisfatte e l'inoltro delle risposte. Inoltre, la fase di aggregazione deve monitorare l'utilizzo delle risorse utilizzate durante la sua esecuzione per evitare carichi di lavoro eccessivi del processore.

Per capire meglio quali sono le componenti di costo da considerare in fase di aggregazione per massimizzare il guadagno della fase di interroga-

zione, si consideri uno scenario in cui sono presenti due nodi SUN ed un nodo CUN. I nodi SUN generano ed inoltrano le query al nodo CUN il quale eseguirà l'algoritmo di aggregazione prima di effettuare il confronto con i dati. In tale scenario, il CUN interpreta il ruolo del server mentre i SUN quello dei client; si osservi che, allo stesso modo, il CUN interpreta il ruolo del client quando inoltra le query ricevute dai SUN al nodo BN il quale, in questo caso, interpreta il ruolo del server.

Per massimizzare il guadagno, la fase di aggregazione deve considerare le seguenti componenti di costo:

1. Server: considera il carico di lavoro che il nodo (nel nostro scenario il CUN) deve effettuare per interrogare l'insieme delle query memorizzate all'arrivo di un nuovo dato. Si definisce quindi

$$cost_{server} \propto size(agg(A)).$$

2. Rete: considera il numero di volte che il dato ricevuto viene inoltrato come risposta e dipende dal numero di query soddisfatte dal dato stesso. Si definisce quindi

$$cost_{network} \propto size(risp)$$

con $risp$ l'insieme dei dati inviati.

3. Client: come per la componente di rete, il nodo che ha inviato/inoltrato la query potrebbe ricevere più dati se l'aggregazione introduce molti dati spuri; si noti che, come è stato definito precedentemente, per questa tesi l'esecuzione della procedura di estrazione non è da considerarsi un costo ulteriore. Si definisce quindi

$$cost_{client} \propto size(risp).$$

Si definisce la seguente funzione chiamata *guadagno di banda* (G_B):

$$G_B = \frac{\sum_{j=1}^{n} size(risp(q_j))}{\sum_{i=1}^{m} size(risp(agg(A_i)))}, con\ q_j \in Q$$

dove viene considerato il fatto che le risposte ad una query aggregata possono essere inviate a nodi differenti.

In particolare, per motivi che saranno noti più avanti, si focalizza l'attenzione sull'aggregazione di una coppia di query; perciò, la funzione diventa:

$$G_B = \frac{size(risp(q_1)) + size(risp(q_2))}{size(risp(q_a))}.$$

In questo caso il risultato della funzione G_B è compreso tra 0 (escluso) e 2. In particolare quando il risultato è:

- $G_B = 2$: le due query sono identiche e sono state inviate da un unico nodo; in questo caso il traffico dati è dimezzato. Si osservi che questo caso, seppur ideale, può verificarsi quando due query uguali generate da nodi diversi vengono ricevute da un unico nodo ed inoltrate verso un altro; quest'ultimo avrà ricevuto due query identiche dallo stesso nodo.

- $1 < G_B < 2$: le query hanno una forte sovrapposizione e il traffico dati risulta ridotto.

- $G_B = 1$: l'aggregazione delle due query introduce un traffico spurio pari al traffico dati risparmiato dalle richieste comuni; in questo caso la banda rimane invariata.

- $0 < G_B < 1$: il traffico dati risulta incrementato in quanto il traffico spurio introdotto è maggiore del traffico risparmiato dalle richieste comuni; si noti che potrebbe comunque convenire aggregare le due query per ridurre il $cost_{server}$.

La dimensione dell'insieme dei dati richiesti, versione dei dati a parte, può essere stimata accuratamente se si assume che ogni dato richiesto esista e se si conosce il dominio dei dati stessi. Nella presente tesi sono stati considerati dati composti da due attributi e query che specificano valori per entrambe gli attributi; ipotizzando di avere la query $q_1 = attr1 : \{val1, val2\}$ && $attr2 : \{val3, val4\}$, il numero di dati richiesti è 4 ed essi sono: $\{(val1; val3), (val1; val4), (val2; val3), (val2; val4)\}$.

A questo punto della trattazione non si è ancora in grado di stimare il $cost_{server}$; nel capitolo successivo saranno effettuati dei test al fine di valutare l'effettivo lavoro computazionale del server al variare di diversi parametri. Quindi, a questo livello, si definisce la funzione costo come la tendenza ad aggregare le query per ridurre il $cost_{server}$ considerando la funzione G_B.

4.2.3 Algoritmi per l'aggregazione

Il problema dell'aggregazione delle query è un problema NP-hard (*Non-deterministic Polynomial-time hard*) ovvero, secondo la teoria della complessità, appartiene ad una classe di problemi non meno difficili di quelli più difficili appartenenti alla stessa classe NP, detti NP-completi. A. Crespo ed altri dimostrano tale assunzione in **[Cre03]**. Per questo motivo sarà brevemente illustrato l'algoritmo esaustivo per poi presentare gli algoritmi euristici.

Algoritmo esaustivo

La figura 4.7 presenta un approccio esaustivo per risolvere il problema dell'aggregazione delle query.

In sostanza, bisogna considerare l'insieme delle parti $S(Q)$ di Q, cioè l'insieme $S(Q)$ di tutti i possibili sottoinsiemi di Q. Da questo bisogna considerare l'insieme delle parti $S(S(Q))$ di $S(Q)$ e filtrare gli elementi che non coprono totalmente Q.

Input : $Q = \{q_1, q_2, ..., q_m\}$.
Output : Optimal solution for the query merging problem.

Generate $\mathbb{S}(Q)$, the superset of the set Q.
Generate $\mathbb{S}(\mathbb{S}(Q))$, the superset of $\mathbb{S}(Q)$.
$A = \{x \mid x \in \mathbb{S}(\mathbb{S}(Q))$ and x is a total cover of $Q\}$
Evaluate the cost of each element of A using the cost model.
The element of A with the minimum cost is the optimal solution.

Figura 4.7: L'algoritmo esaustivo **[Cre03]**.

Un esempio esplicativo è mostrato in Figura 4.8.

Q = {a,b}

S(Q) = {{Ø}, {a}, {b}, {a,b}}

S(S(Q)) = {{Ø}, {{a}}, {{b}}, {{a,b}}, {{a}, {b}}, {{a}, {a,b}}, {{b}, {a,b}}}

Figura 4.8: Un esempio di insieme delle parti con le soluzioni ammissibili sottolineate.

Per i soli elementi risultanti dall'operazione di filtraggio dev'essere valutata la funzione costo; la soluzione di costo inferiore è la soluzione ottima. La complessità computazione di tale algoritmo è doppiamente esponenziale rispetto al numero delle query $O(2^{2^{|Q|}})$, poiché il primo passo genera un insieme $S(Q)$ di $2^{|Q|}$ elementi, mentre il secondo passo ne genera un totale di $2^{2^{|Q|}}$. Si può facilmente giungere alla conclusione che un algoritmo esaustivo non è utilizzabile se non per un $|Q|$ molto piccolo.

Algoritmi euristici

Gli algoritmi euristici, invece, consentono di ottenere una 'buona' soluzione con una complessità computazionale limitata, più adatti quindi al lavoro di questa tesi. In **[Cre03]** viene proposto un algoritmo euristico di base il quale può essere esteso per migliorarne le performance e la bontà della soluzione. L'algoritmo principale viene chiamato 'Algoritmo di Aggregazione di Coppia' e si fonda su due assunzioni: il processo di decisione dell'aggregazione basato sul confronto tra coppie e la proprietà di singola allocazione. La prima non può definirsi corretta in generale, ma consente efficientemente di ottenere soluzioni molto vicine alla soluzione ottima. La seconda assume che nella soluzione ottima non ci possono essere ripetizioni della stessa query nei vari sottoinsiemi, quindi una query può non essere aggregata oppure essere aggregata con un unico sottoinsieme; la Figura 4.9 ne riporta un esempio da confrontarsi con la Figura 4.8.

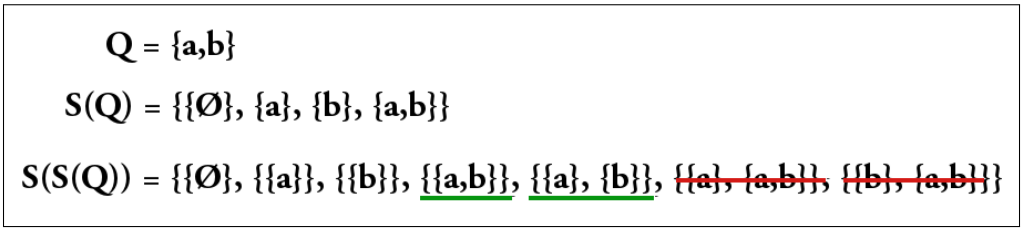

Figura 4.9: Un esempio di insieme delle parti considerando la proprietà di singola allocazione.

L'algoritmo di aggregazione di coppia considera in ingresso un insieme di insiemi di query, inizialmente tutti contenenti un'unica query. Per tutte le coppie di insiemi l'algoritmo elabora la variazione di costo che si avrebbe aggregando i due insiemi e, al termine di tale processo, aggrega la coppia che introduce la più alta (in valore assoluto) variazione negativa. Al ciclo successivo si riconsiderano gli stessi insiemi considerati nel precedente ciclo esclusi quelli aggregati e aggiungendo l'insieme frutto dell'aggregazione. L'algoritmo termina quando non vi è più nessuna coppia di insieme che introduce una variazione negativa di costo. La comples-

sità dell'algoritmo risulta essere $O(|Q|^2)$ per via del confronto tra coppie di query.

Una debolezza di tale approccio sta nel fatto di poter rimanere intrappolati in minimi locali della funzione costo e non trovare il minimo globale. In generale, non esiste alcun algoritmo polinomiale che possa eliminare tale debolezza ma la variante chiamata 'Algoritmo di Ricerca Guidata' può apportare miglioramenti notevoli. Nello specifico, ad ogni passo dell'algoritmo uno degli insiemi di insiemi può essere suddiviso in due parti: una contenente solo una query e l'altra contenente il resto. Si consideri ad esempio l'insieme $\{\{a\},\{b,c\},\{d,e\}\}$ che può essere trasformato in $\{a\},\{b\},\{c\},\{d,e\}$ se nel precedente ciclo è stato aggregato d con e. L'algoritmo viene riportato in Figura 4.10.

Input : $Q = \{q_1, q_2, ..., q_m\}$.
Output : Solution for the query merging problem.
Create set $\mathcal{T}$ of initial states.
For each initial state $\mathcal{S}$ in $\mathcal{T}$:
    $\mathcal{M} = \mathcal{S}$
    Repeat:
        For each pair of sets M_i, M_j in $\mathcal{M}$:
            Compute the cost of $(\mathcal{M} - M_i - M_j) \cup (\{M_i \cup M_j\})$.
        For each query q of each set S in $\mathcal{M}$:
            Compute the cost of $(\mathcal{M} - S) \cup \{S - \{q\})\} \cup \{\{q\}\}$.
        Perform the operation that reduces the total cost the most.
    Until no beneficial operation is possible.

Figura 4.10: L'algoritmo di ricerca guidata **[Cre03]**.

In questo modo è possibile riconsiderare delle 'cattive' scelte effettuate nei precedenti cicli. In alternativa si possono utilizzare diversi stati iniziali e scegliere quello che porta al costo minimo. In questo caso la complessità dell'algoritmo cresce e diventa $O(|T| \cdot |Q|^{|Q|})$, con $|T|$ il numero di stati iniziali considerati.

Per migliorare le performance dell'algoritmo, valido per tutte le versioni, è possibile utilizzare 'l'Algoritmo di Clustering' che utilizza un approccio 'divide et impera' per il problema di aggregazione delle query. Le fondamenta di tale algoritmo di basano sulla definizione di una metrica di 'distanza' tra le query. Se la distanza tra due query è 'abbastanza grande', si possono ignorare tutte le possibili combinazioni di query aggregate che le contengono. La Figura 4.11 presenta un'intuizione grafica di tale approccio. In essa ci sono cinque query, da q_1 a q_5. Le query q_1, q_3 e q_5

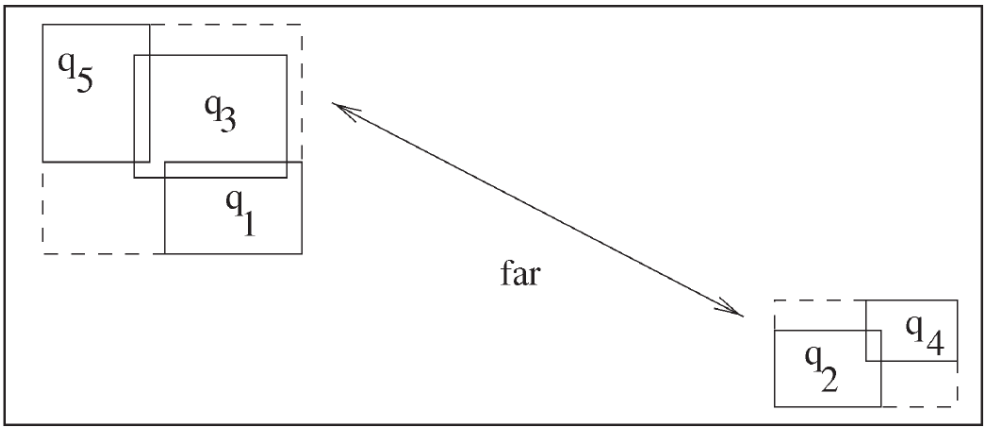

Figura 4.11: Un esempio di clusterizzazione.

sono molto simili e perciò sono dei buoni candidati per essere aggregati insieme. Le query q_2 e q_4 sono invece distanti dalle precedenti e potrebbe non avere senso considerare l'aggregazione con le prime. In particolare, l'algoritmo di clustering elabora la 'distanza' tra ogni coppia di query e se questa distanza è sotto una certa soglia, la coppia viene inserita all'interno dello stesso cluster. Si osservi inoltre che vale la proprietà transitiva: se q_1 è vicina a q_3 e q_3 è vicina a q_5 ma q_1 e q_5 non sono vicine abbastanza, le tre query sono comunque inserite nello stesso cluster. La complessità di tale algoritmo è $O(|Q|^2)$.

L'esecuzione dell'algoritmo di clustering suddivide l'insieme di partenza in diversi sotto-insiemi per i quali bisogna eseguire in maniera totalmente indipendente l'algoritmo di aggregazione preferito, aggregazione di coppia o ricerca guidata. In questo modo si riduce la dimensione del

problema e quindi il lavoro totale del processore.

In questo capitolo sono stati mostrati i concetti fondamentali delle tecniche di sintetizzazione e di aggregazione delle query. Come si è visto, le tecniche di aggregazione ben si prestano all'utilizzo nei sistemi context-aware poiché applicabili in modo differenziato sui nodi eterogenei che partecipano alla distribuzione di contesto. Per tale motivo, nello sviluppo di questa tesi si è deciso di progettare una soluzione basata su queste ultime.

Capitolo 5

Progetto ed implementazione della soluzione

Dopo aver definito i sistemi context-aware e le problematiche ad essi associate, questo capitolo illustra la soluzione proposta in questa tesi.

Il Paragrafo 5.1 riporta l'analisi dei requisiti per comprendere meglio il comportamento atteso dalla soluzione. Nel Paragrafo 5.2 si riportano alcune informazioni sui componenti di SALES che sono stati utilizzati e modificati. La soluzione proposta consente di differenziare il comportamento tra il caso di dispositivi a risorse scarse e il caso di dispositivi dotati di elevata capacità computazionale; le descrizioni di entrambe le soluzioni vengono presentate rispettivamente nel Paragrafo 5.3 e nel Paragrafo 5.4. Infine, nel Paragrafo 5.5 si riporta l'implementazione di una tecnica per distribuire uniformemente nel tempo i picchi di utilizzo del processore.

5.1 Analisi dei requisiti

Partendo dal dato di fatto che in un sistema context-aware le query emesse da nodi che si trovano nella stessa area geografica presentano di solito similarità e coperture parziali o totali, l'obiettivo di questa tesi è quello di evidenziare e sfruttare tali similarità per modulare opportunamente il carico di lavoro dei singoli nodi e il traffico dei dati di contesto genera-

to in risposta alle richieste. In particolare, si richiede di progettare ed implementare una soluzione sull'infrastruttura SALES per consentire il raggiungimento di tali obiettivi.

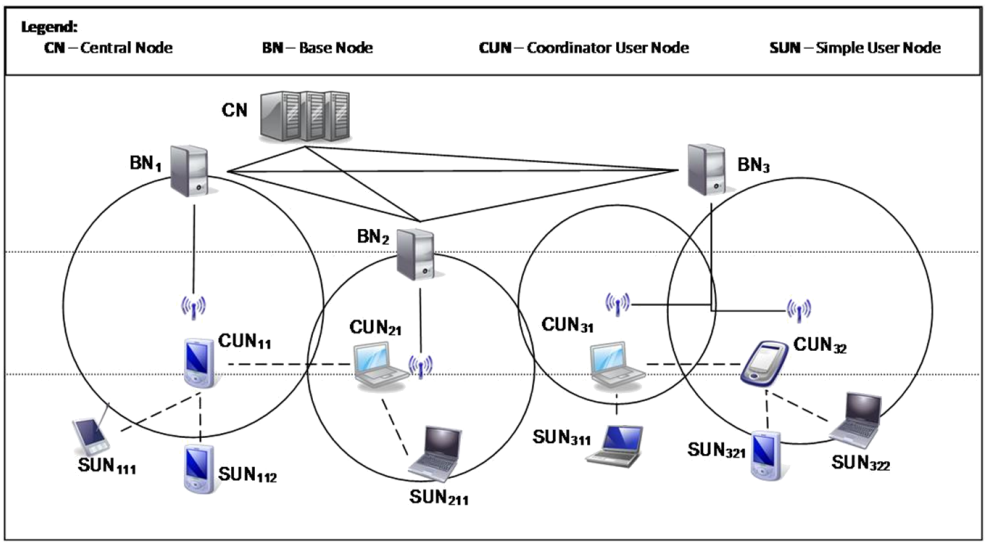

Figura 5.1: Architettura distribuita di SALES.

Nel middleware SALES, del quale la Figura 5.1 ne riporta l'architettura distribuita, i nodi che generano i dati e le richieste di contesto sono tipicamente quelli appartenenti all'infrastruttura wireless mobile.

Tuttavia, tutti i nodi partecipano attivamente alla distribuzione dei dati di contesto, sia che essi appartengano all'infrastruttura mobile oppure a quella fissa. Per questo motivo si ritiene di dover realizzare una soluzione che moduli il proprio comportamento per adattarsi alle caratteristiche computazionali del dispositivo sul quale esegue. Risulta altresì evidente che mentre sui dispositivi dotati di scarse risorse computazionali assume maggiore importanza l'obiettivo di riduzione del carico di lavoro, per gli altri dispositivi si ritiene più importante ridurre il traffico dei dati di contesto introdotto dalle risposte. Limitando il traffico si può ottenere un duplice effetto: minore occupazione di banda e riduzione ulteriore del carico di lavoro dei nodi a risorse scarse, grazie alla diminuzione del numero di duplicati delle risposte inoltrate dall'infrastruttura fissa verso quella mobile. Per questo motivo si prevede, sin dalla fase di analisi, la realizzazione di

una soluzione che implementi comportamenti diversi in base alle capacità computazionali del nodo che li attua.

Nella soluzione proposta in questa tesi, ogni nodo che prende parte al processo di distribuzione dei dati di contesto deve eseguire l'algoritmo di aggregazione. In dettaglio, quando un nodo riceve una nuova query deve memorizzarla ed inoltrarla allo stesso modo e con gli stessi tempi della versione originale di SALES. Questa scelta è motivata dal fatto che la disseminazione della query non può essere ritardata a causa del processo di aggregazione altrimenti si ritarderebbe l'invio delle risposte già presenti. Quando un nodo riceve una query, ne memorizza una copia e la inoltra agli altri nodi secondo il processo di distribuzione di SALES. Dopo aver ricevuto un certo numero di query, il nodo può decidere di eseguire l'algoritmo di aggregazione sulle copie memorizzate.

Per capire il funzionamento della soluzione, si consideri a titolo di esempio una parte dell'architettura SALES e le query geografiche entrambi presentate in Figura 5.2. Si specifica che l'utilizzo delle query geografi-

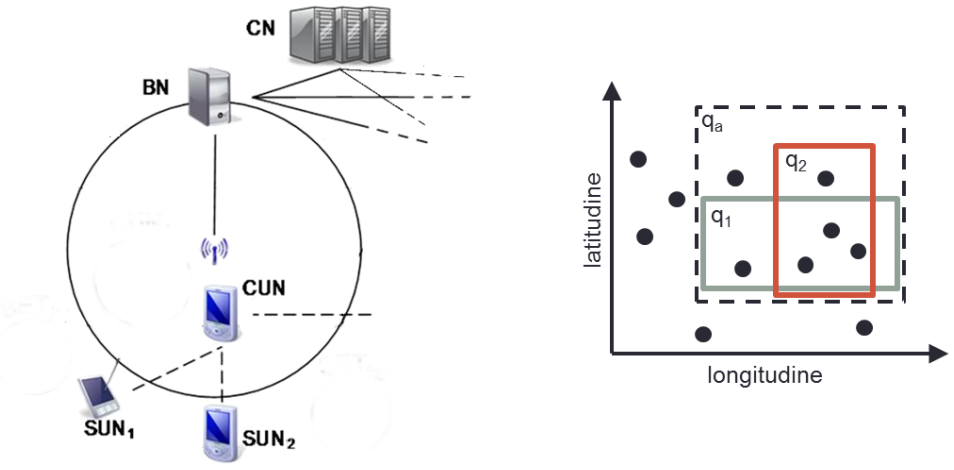

Figura 5.2: Funzionamento base di SALES.

che negli esempi riportati ha il solo scopo di rendere più facilmente comprensibile la trattazione. Si supponga che i nodi SUN_1 e SUN_2 generino

rispettivamente le query q_1 e q_2. Il processo di distribuzione delle query in SALES prevede che ogni nodo effettui prima una distribuzione orizzontale e poi una distribuzione verticale, perciò, considerando unicamente la query q_1, esso si articola nei seguenti passi:

- il SUN_1 invia la query al CUN
- il CUN verifica la presenza di dati locali che soddisfano la query e, in caso positivo, ne programma l'invio; poi memorizza la query e l'invia prima orizzontalmente, se sono presenti nodi orizzontali, e poi verticalmente al BN
- il BN, allo stesso modo, verifica il proprio insieme di dati, memorizza la query e l'inoltra prima orizzontalmente e poi verticalmente al CN
- CN, dopo aver verificato il proprio insieme di dati, memorizza la query.

Lo stato finale dei nodi viene proposto in Figura 5.3 dove vengono generate le query q_1 e q_2, rispettivamente dai nodi SUN_1 e SUN_2. Il cerchio vicino ad ogni nodo rappresenta l'insieme delle query ricevute e memorizzate dal nodo stesso. Nella versione originale i nodi non effettuano nessuna elaborazione sulle query ricevute perciò, ipotizzando che BN riceva un dato che soddisfi entrambe le query q_1 e q_2, vengono generati i seguenti passi:

- il BN riceve il dato, verifica il soddisfacimento della query q_1 e lo inoltra al nodo CUN
- il CUN riceve il dato, verifica a sua volta il soddisfacimento della query q_1 e lo inoltra al nodo SUN_1
- il CUN verifica che il dato soddisfi anche la query q_2 e lo inoltra al nodo SUN_2
- il BN verifica che il dato ricevuto soddisfi la query q_2 e lo inoltra nuovamente al nodo CUN

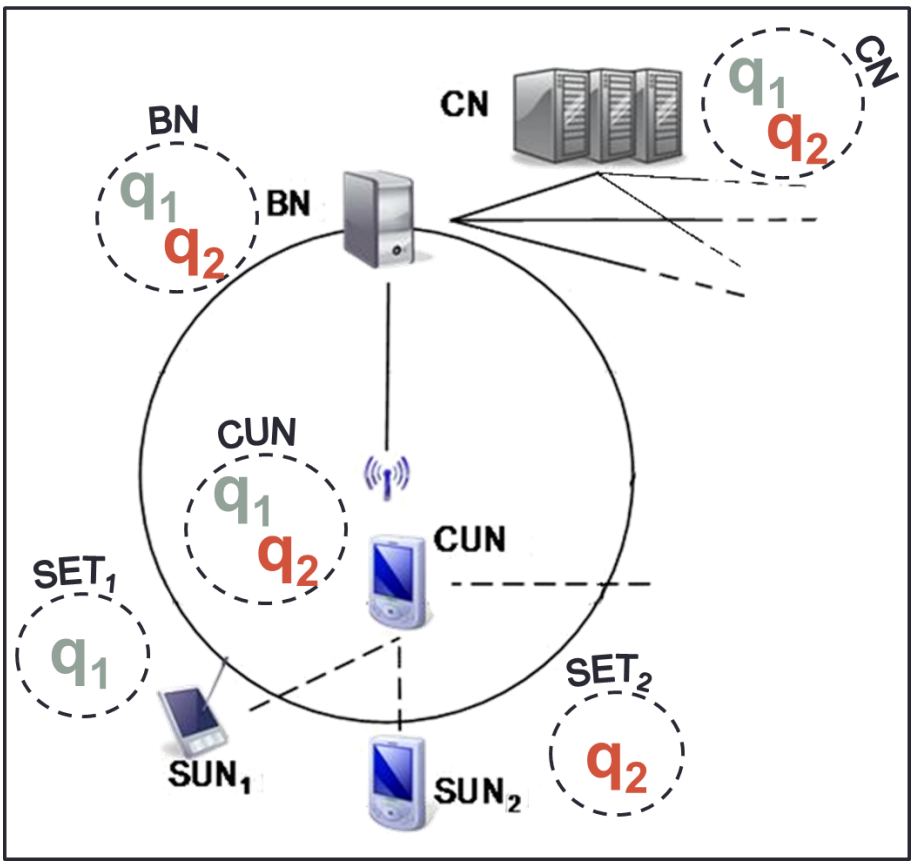

Figura 5.3: Esempio di diffusione delle query in SALES.

- il CUN riceve nuovamente lo stesso dato e, ipotizzando che sia dotato di una *cache* dei dati ricevuti, lo riconosce e lo scarta.

Tali passi mettono in mostra come, ipotizzando la presenza di una cache, i nodi BN e CUN devono verificare due volte la corrispondenza tra dato e query, una volta per q_1 e l'altra per q_2. Inoltre il dato viene inviato due volte dal BN al CUN comportando un consumo duplice della banda. Si osservi che alcuni passi effettuati da nodi diversi possono essere eseguiti in parallelo.

La soluzione proposta in questa tesi prevede che le query ricevute siano aggregate, se ritenuto opportuno, in una nuova query q_a la quale viene generata dal rilassamento delle query originali come mostra la Figura 5.2 presentata precedentemente.

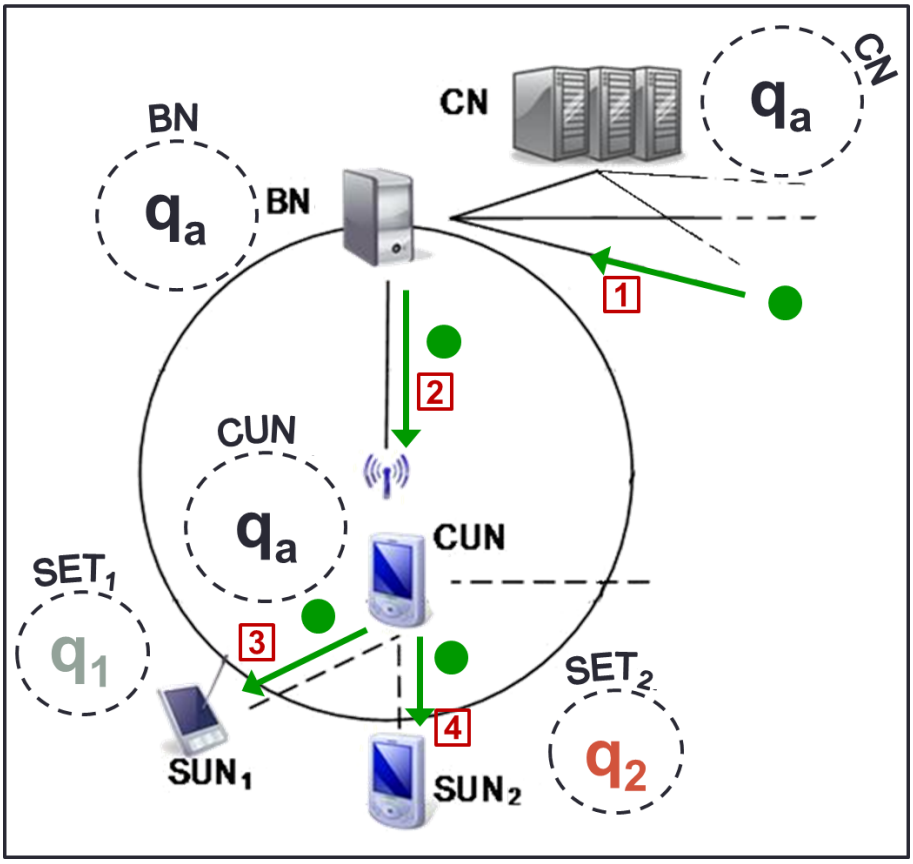

Figura 5.4: Esempio di inoltro delle risposte in SALES.

In questo caso, come mostra la Figura 5.4, i passi generati alla ricezione di un dato che soddisfa entrambe le query originali q_1 e q_2 sono:

- il BN riceve il dato, verifica il soddisfacimento della query q_a e lo inoltra al nodo CUN
- il CUN riceve il dato, verifica a sua volta il soddisfacimento della query q_a e lo inoltra ai nodi SUN_1 e SUN_2

I nodi BN e CUN, a differenza del comportamento assunto nella versione originale di SALES, verificano un'unica volta la corrispondenza tra dato e query in quanto l'insieme delle query ricevute contiene solo la query q_a. Inoltre il dato viene inviato dal BN al CUN un'unica volta.

Si osservi che quanto appena presentato è il caso favorevole in cui il dato ricevuto soddisfa entrambe le query originali. In alcuni casi si prevede di ricevere dati che soddisfano un'unica query oppure nessuna query, ovvero dati spuri.

5.2 Componenti di SALES coinvolti

Questo paragrafo descrive alcune delle classi utilizzate e adattate in questa tesi per poter comprendere gli spunti d'implementazione proposti. Le principali classi d'interesse sono: $ContextQuery$, $ContextData-Filter$, $MembershipTestFilter$, $ORFilterCollection$ e $ANDFilterCollection$. Si riporta il diagramma delle classi in Figura 5.5 per evidenziare le relazioni statiche.

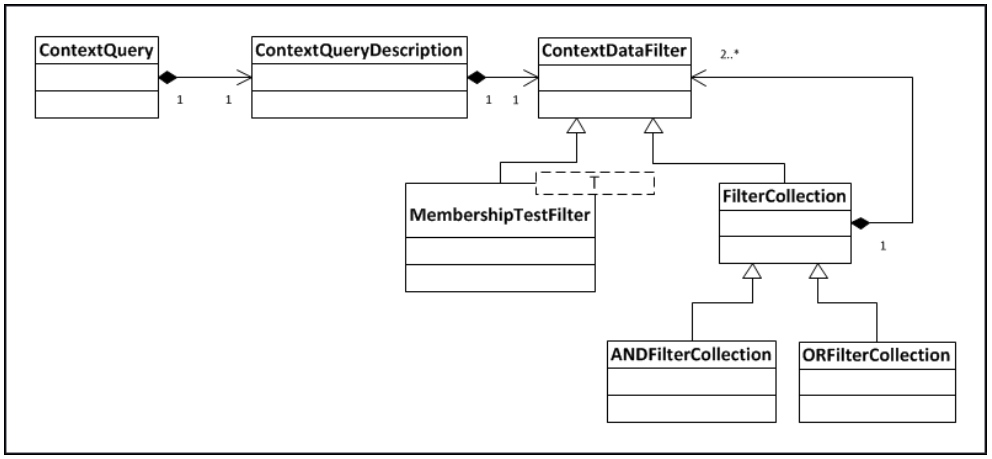

Figura 5.5: Diagramma delle classi di principale interesse.

La classe $ContextQuery$ implementa una query di contesto ed è composta dalla classe $ContextDataFilter$. Questa è una classe astratta che rappresenta il filtro della query ed è estesa dalla classe concreta $Membership-TestFilter$ e dalla classe astratta $FilterCollection$ la quale a sua volta è estesa dalle classi concrete $ORFilterCollection$ e $ANDFilterCollection$. La classe $MembershipTestFilter$, come già anticipato nel Paragrafo 3.6, implementa un filtro su uno degli attributi dei dati ed è il principale filtro

utilizzato per i test della tesi. Le classi $ORFilterCollection$ e $ANDFilter-Collection$ implementano rispettivamente l'operatore logico OR e AND.

5.3 Soluzione per dispositivi a risorse scarse

Nei sistemi context-aware i nodi mobili sono spesso dei dispositivi a risorse scarse, ovvero macchine dotate di una limitata capacità computazionale; ne sono un esempio PDA e telefoni cellulari. In quest'ottica si è costruita una soluzione in grado di alleggerire il carico di lavoro durante la ricezione dei dati.

La soluzione proposta per dispositivi mobili si articola in quattro fasi:

1. modulazione del carico di lavoro del processore
2. partizionamento dell'insieme delle query ricevute
3. calcolo della soglia per modulare il processo di aggregazione
4. esecuzione dell'algoritmo di aggregazione per ogni cluster.

Per facilitare la comprensione saranno trattate dapprima le fasi 2 e 4 e poi le fasi 3 e 4 in quanto più di dettaglio.

5.3.1 Partizionamento

La fase di partizionamento ha lo scopo di spezzare la dimensione del problema identificando e separando gli insiemi di query che sicuramente non saranno aggregate nella fase 4. Per poter prendere tale decisione bisogna definire una *funzione distanza* che consente di decidere se due query sono sufficientemente distanti da essere considerate non aggregabili. La funzione distanza è stata definita nella seguente maniera: se due query fanno richiesta di almeno un dato in comune, allora vengono inserite nello stesso cluster; altrimenti le due query saranno inserite in cluster differenti. Come

mostra la Figura 5.6, vale la proprietà transitiva: se due query sono lontane ma esiste una terza query vicina ad entrambe, allora tutte e tre le query saranno inserite nello stesso cluster.

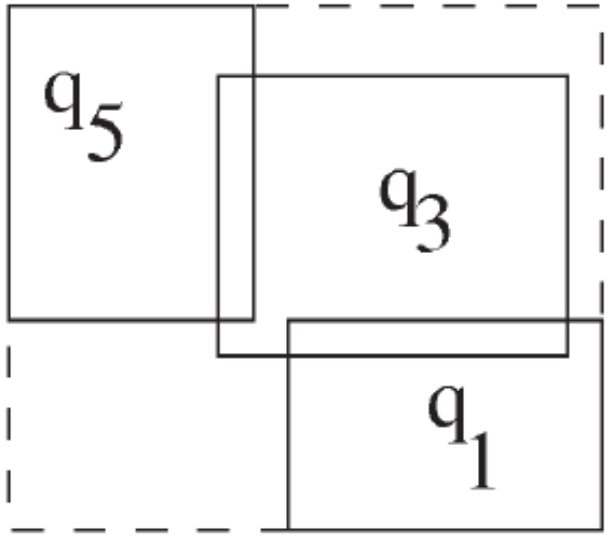

Figura 5.6: Proprietà transitiva dell'operazione di partizionamento.

L'operazione di partizionamento prevede in ingresso un insieme di query e restituisce in uscita un insieme di cluster, ognuno dei quali contiene le query candidate per l'aggregazione. L'aggregazione di query appartenenti a cluster differenti non deve essere presa in considerazione.

Il Listato 5.1 mostra una semplificazione del metodo '*clustering*'.

Listing 5.1: Semplificazione del metodo di partizionamento.

```
protected List<Cluster> clustering(List<ContextQuery> listOfQueries) {
  List<Cluster> listOfClusters = new ArrayList<Cluster>();
  List<Cluster> clustersToMerge = null;
  boolean match;

  // per ogni query ricevuta
  for (ContextQuery newQuery : listOfQueries) {
    clustersToMerge = new ArrayList<Cluster>();
    match = false;
```

```
    // verifica la sovrapposizione con le query già esaminate
    for (Cluster cluster : listOfClusters)
      for (ContextQuery oldQuery : cluster.getQueries())
        if (newQuery.getFilter().match(oldQuery.getFilter())) {
          clustersToMerge.add(cluster);
          match = true;
          break;
        }

    // se c'è almeno una sovrapposizione con una query già inserita
    if (match) {
      Cluster mainCluster = clustersToMerge.get(0);

      // se ci sono più sovrapposizioni
      //  -> riunisci tutte le query in un unico cluster
      if (clustersToMerge.size() > 1)
        for (int i = 1; i < clustersToMerge.size(); i++) {
          Cluster clusterToRemove = clustersToMerge.get(i);
          for (ContextQuery queryToMove : clusterToRemove.getQueries())
            mainCluster.add(queryToMove);

          listOfClusters.remove(clusterToRemove);
        }
      mainCluster.add(newQuery);
    }
    // altrimenti crea un nuovo cluster
    else {
      Cluster cluster = new Cluster();
      cluster.add(newQuery);
      listOfClusters.add(cluster);
    }
  }
  return listOfClusters;
}
```

Per verificare se due query si sovrappongono è stata implementato il metodo '*match*' all'interno delle classi che implementano i filtri utilizzati nella tesi: il *MembershipTestFilter* (MTF) e i filtri che permettono di combinare logicamente diversi MTF, ovvero *ANDFilterCollection* (ANDFC) ed *ORFilterCollection* (ORFC). Di seguito si illustrano i principali casi di verifica delle sovrapposizioni:

1. quando entrambe le query sono formate da un MTF: sono sovrapposte se riguardano attributi differenti oppure riguardano lo stesso attributo e l'intersezione degli insiemi dei valori possibili è non nulla

2. quando una query è formata da un ANDFC tra due MTF e l'altra da un MTF: sono sovrapposte se entrambe i due MTF della query ANDFC soddisfano il punto 1 con la query MTF

3. quando una query è formata da un ORFC tra due MTF e l'altra da un MTF: sono sovrapposte se almeno un MTF dell'ORFC soddisfa il punto 1 con la query MTF

4. quando entrambe le query sono formate da un ANDFC tra due MTF: sono sovrapposte se entrambe i due MTF di una query soddisfano il punto 2 con l'altra query

5. quando una query è formata da un OR tra due MTF e l'altra da un ANDFC tra due MTF: sono sovrapposte se almeno uno dei due MTF della query ORFC soddisfa il punto 2 con la query ANDFC

6. quando entrambe le query sono formate da un ORFC tra due MTF: sono sovrapposte se almeno uno dei due MTF della prima query soddisfa il punto 3 con la seconda query.

Il Listato 5.2 riporta, per motivi di spazio, solo il caso 1.

Listing 5.2: Il metodo che verifica la sovrapposizione tra due filtri MTF.

```
public boolean match(ContextDataFilter filter) {
  if (filter.getFilterType() == FilterType.PLAIN_MEMBERSHIP_FILTER) {
    MembershipTestFilter<T> f = (MembershipTestFilter<T>) filter;
    if (this.getAttributeName().equals(f.getAttributeName())) {
      for (T value : this.values)
        if (f.values.contains(value))
          return true;

      return false;
    } else
      return true;
  }
}
```

Per ogni cluster identificato dalla fase di partizionamento viene avviato separatamente l'algoritmo di aggregazione, riducendo così la dimensione del problema e il lavoro complessivo del processore.

5.3.2 Algoritmo di aggregazione

Si è deciso di implementare l'*algoritmo di aggregazione di coppia*, brevemente esposto nel Sottoparagrafo 4.2.3. La funzione di costo utilizzata è quella riportata nel Sottoparagrafo 4.2.2 definita con il nome di *Guadagno di banda*.

Per completezza, nel Listato 5.3 si riporta una semplificazione del codice del metodo '*gainBandwidth*' che implementa la funzione di costo.

Listing 5.3: Semplificazione del metodo che esegue la funzione *guadagno di banda*.

```
private static float gainBandwidth(ContextQuery q1, ContextQuery q2) {
  ContextDataFilter f1 = q1.getFilter().copy();
  ContextDataFilter f2 = q2.getFilter().copy();
  float numAnsSep = f1.NumOfAnswers() + f2.NumOfAnswers();
  f1.merge(f2);
  float numAnsMer = f1.NumOfAnswers()* relaysSize(q1, q2);
```

```
    return numAnsSep / numAnsMer;
}
```

Questo metodo chiama al suo interno un altro metodo, '*merge*', il quale comportamento sarà illustrato successivamente.

Dopo aver implementato la funzione G_B si riporta il seguente concetto dell'algebra booleana: qualsiasi funzione booleana può essere espressa in forma canonica come somma di prodotto (OR di AND) e prodotto di somme (AND di OR). Grazie ad esso è stato possibile considerare un insieme limitato di modelli di query in grado di rappresentare qualsiasi espressione logica booleana. Tale insieme comprende:

- unico MTF che associa ad un attributo un insieme di valori possibili
- ANDFC di MTF, ognuno formato come descritto nel precedente punto e con attributi differenti
- ORFC dei precedenti modelli.

Si è poi passati all'implementazione dell'algoritmo di aggregazione vero e proprio che da un lato definisce come avviene il processo di aggregazione e dall'altro consente di decidere quando scatenare tale processo. Il processo di aggregazione prevede il rilassamento delle query da aggregare. Di seguito si riportano le descrizioni dei risultati elaborati considerando in ingresso i principali modelli di query:

1. due query MTF: assumendo che abbiano lo stesso attributo, la query risultante sarà un MTF con l'insieme dei possibili valori derivanti dall'unione degli insiemi dei valori delle query MTF di partenza;
2. una query ANDFC di MTF e una query MTF: assumendo che uno ed un solo MTF della query ANDFC abbia lo stesso attributo della query MTF, la query risultante sarà il MTF risultante dall'aggregazione descritta al punto 1 tra la query MTF e l'MTF della query ANDFC con lo stesso attributo;

3. una query ORFC di MTF e una query MTF: assumendo che uno ed un solo MTF della query ORFC abbia lo stesso attributo della query MTF, la query risultante sarà la query ORFC di partenza il quale MTF con lo stesso attributo della query MTF viene sostituito dal risultato dell'aggregazione descritta al punto 1 tra il MTF della query ORFC e la query MTF;

4. due query ANDFC di MTF: la query risultante sarà un ANDFC formato da un numero di MTF pari a quanti MTF con lo stesso attributo hanno in comune le due query di partenza. Ogni MTF della query risultante sarà il risultato dell'aggregazione descritta al punto 1 tra i MTF con lo stesso attributo. Se l'ANDFC risultante ha un unico MTF, la query risultante sarà il MTF stesso.

5. due query ORFC di MTF: la query risultante sarà un ORFC formato da tutti gli MTF delle query di partenza. Se ci sono MTF con lo stesso attributo, essi saranno aggregati come descritto nel punto 1;

6. una query ORFC di MTF e una query ANDFC di MTF: la query risultante sarà un ORFC lo stesso numero di MTF della query ORFC partenza. Ognuno di questi MTF, se hanno lo stesso attributo di uno degli MTF della query ANDFC, saranno aggregati con essa come descritto nel punto 2;

7. una query ORFC di ANDFC di MTF e qualunque altra query: dapprima si cerca di aggregare tra loro gli ANDFC della query ORFC come descritto nel punto 4. Il risultato (o i risultati) di tale operazione sarà aggregato come descritto nei punti precedenti in base all'altra query di partenza considerata.

Per facilitare la comprensione, in Figura 5.7 vengono proposti due esempi di aggregazione: in alto il caso descritto al punto 1 dell'elenco e in basso il caso descritto al punto 2.

Per decidere quando aggregare, l'algoritmo valuta ciclicamente la funzione G_B per tutte le coppie di query ed elegge quella con il miglior risulta-

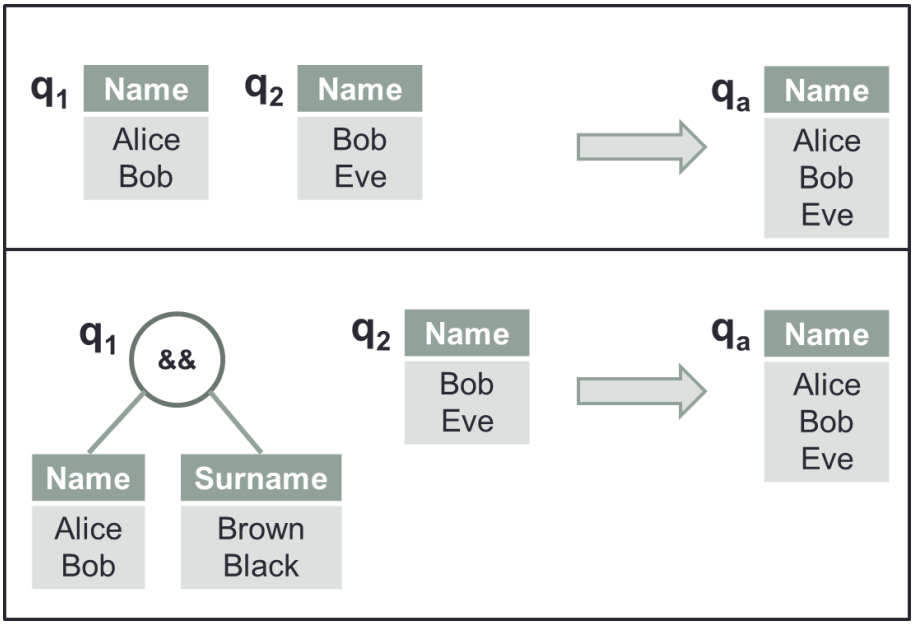

Figura 5.7: Esempi di aggregazione delle query in SALES.

to, ovvero il risultato più alto; tale scelta rispecchia la volontà di aggregare la coppia di query che più riduce il traffico dati, o introduce meno traffico spurio. Come accennato nel precedente capitolo, un punto debole di questo comportamento sta nel fatto di poter rimanere intrappolati in massimi locali della funzione e non trovare il massimo globale. Per avere una computazione più leggera possibile, si è deciso di non implementare tecniche di riduzione di tale rischio. Dopo aver eletto la coppia con il miglior risultato bisogna valutare se l'aggregazione risulta vantaggiosa oppure no. Per questo motivo si utilizza una soglia, chiamata *soglia di aggregazione*, la quale definizione è presentata nel successivo sottoparagrafo.

Per completezza, nel Listato 5.4 si riporta un estratto semplificato del codice che confronta ciclicamente le query e richiama l'algoritmo di aggregazione.

Listing 5.4: Semplificazione del metodo che confronta ciclicamente le query e richiama l'algoritmo di aggregazione.

```
do {
  float bestScore = 0;
```

```
int bestMatchKey = 0;
Map<Integer, MatchScore> scoreBoard =
        new ConcurrentHashMap<Integer, MatchScore>();
  for (int i = 0; i < listOfQueries.size() − 1; i++) {
    ContextQuery q1 = listOfQueries.get(i);
    for (int j = i + 1; j < listOfQueries.size(); j++) {
      ContextQuery q2 = listOfQueries.get(j);
      MatchScore ms = new MatchScore(q1, q2);
      MatchScore msTemp;
      int hashCode = ms.hashCode();
      int hashCode_temp = 0;
      float score = 0;
      // chiave già presente
      if ((msTemp = scoreBoard.get(hashCode)) != null)
        hashCode_temp = hashCode;
      // chiave non presente
      else {
        msTemp = ms;
        hashCode_temp = hashCode;
        score = costFunction(q1, q2);
        msTemp.setScore(score);
        scoreBoard.put(hashCode_temp, msTemp);
      }
      if (bestScore < msTemp.getScore()) {
        bestScore = score;
        bestMatchKey = hashCode_temp;
      }
    }
  }
  if (bestScore == 0.0 || bestScore < this.threshold) {
    break;
  }
  MatchScore bestMatchScore = scoreBoard.get(bestMatchKey);
  ContextQuery q1 = bestMatchScore.getQuery1();
  ContextQuery q2 = bestMatchScore.getQuery2();
```

```
    merge(q1, q2);
    updateScoreBoard(scoreBoard);
} while (bestScore > this.threashold);
```

Ad ogni ciclo molte valutazioni effettuate al ciclo precedente sono ancora valide e quindi non debbono essere ricalcolate; in particolare, sono valide tutte le valutazioni che non coinvolgono le query che sono state aggregate. Per non ripetere più volte gli stessi confronti, come mostra il Listato 5.4, è stata implementata una tabella per memorizzare il risultato della funzione G_B per ogni coppia di query. Quando si aggregano due query bisogna aggiornare tale tabella eliminando tutti i risultati che si riferiscono ad almeno una delle due query aggregate.

5.3.3 Calcolo della soglia di aggregazione

La soglia da utilizzare nel processo di aggregazione per valutare se due query debbano essere aggregate oppure no dipende dalla somiglianza dell'intero insieme delle query considerate. Per constatare il lavoro fatto dal processore nella fase di interrogazione, definita nel Sottoparagrafo 4.2.2, sono stati eseguiti alcuni test al variare della somiglianza delle query ricevute e della soglia scelta. Le query utilizzate durante i test saranno definite con maggior dettaglio nel capitolo successivo; per adesso ci si limita a dire che le query sono state prodotte secondo una distribuzione gaussiana di $valor\ minimo = 0$, $valor\ massimo = 100$, $media = 50$ e similarità variabile. La misura della similarità viene espressa mediante la deviazione standard con la seguente relazione:

$$similarità = \frac{1}{deviazione\ standard}.$$

Quindi con deviazione standard bassa s'intende un'alta similarità mentre con deviazione standard alta s'intende una bassa similarità.

La Figura 5.8 riporta i risultati sul tempo di utilizzo del processore sommando gli intervalli di tempo in cui il processore esegue i confronti tra i

dati e le query memorizzate. Inoltre tale figura riporta la percentuale media di utilizzo del processore durante tutta la fase di interrogazione. I grafici sono stati costruiti al variare della deviazione standard (sui valori 1, 5, 10, 15 e 20) e utilizzando 3 soglie di aggregazione differenti, ovvero 0, 0.5 e 1.0. Si riporta inoltre, in linea tratteggiata, il caso senza aggregazione.

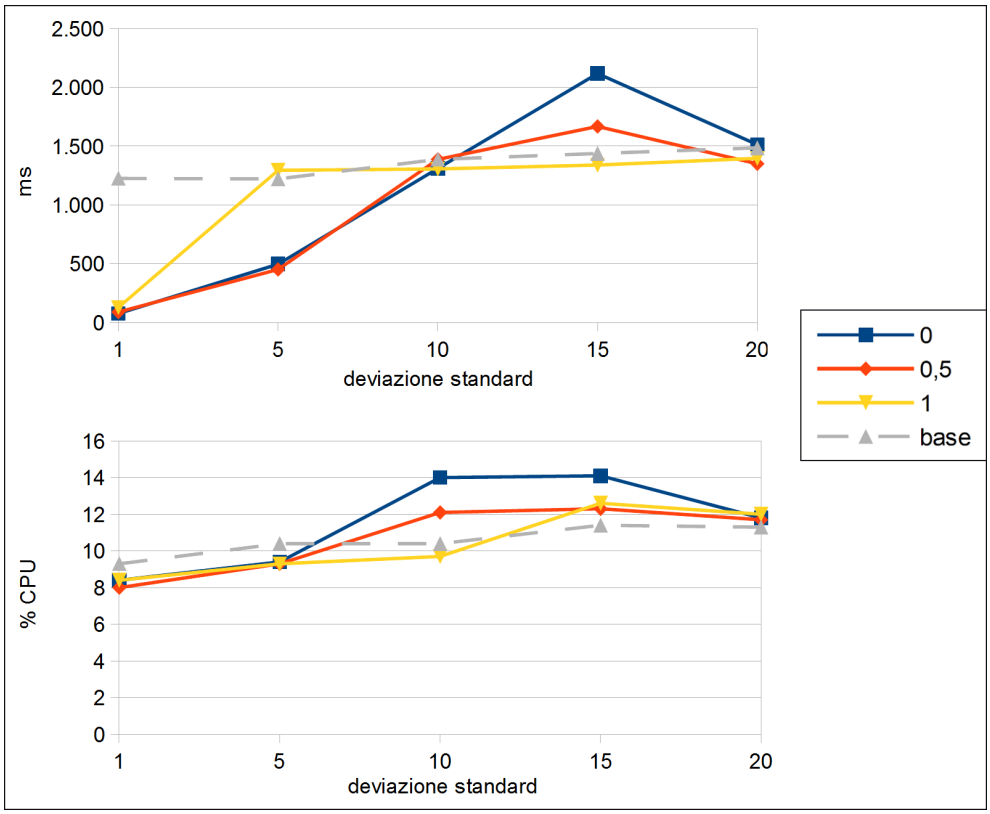

Figura 5.8: Lavoro del processore in fase di interrogazione al variare della somiglianza delle query e della soglia di aggregazione.

Analizzando i risultati si possono fare le seguenti osservazioni:

- l'aggregazione può introdurre vantaggio oppure uno svantaggio poiché per alcune deviazioni standard, se $soglia < 1$, l'utilizzo di CPU è maggiore rispetto al caso base (linea tratteggiata)

- nel caso in cui $soglia = 1$ (linea continua con triangoli) non si ha in nessun caso uno svantaggio in quanto il traffico dati non aumenta e

al limite ci si allinea al caso base quando le query sono troppo diverse per essere aggregate

- nei casi in cui $soglia < 1$ si riesce ad ottenere un vantaggio anche per deviazioni standard più elevate rispetto al caso precedente; questo conferma il fatto che seppur si sta aumentando il traffico e quindi l'invio di dati, il lavoro totale del processore diminuisce

- in prossimità di una deviazione standard pari a 10 e oltre, non conviene aggregare poiché non si introduce un vantaggio o si introduce uno svantaggio

- partendo dalla deviazione standard pari a 5 e andando verso 10, conviene aumentare la soglia per non rischiare di introdurre uno svantaggio.

Da queste osservazioni sono state identificate 3 diverse aree di deviazione standard per le quali è opportuno modulare la soglia di aggregazione in maniera indipendente:

1. se $deviazione\ standard <= 5$: la soglia viene fissata a 0;

2. se $5 < deviazione\ standard <= 10$: la soglia deve essere modulata opportunamente;

3. se $10 < deviazione\ standard$: l'algoritmo di aggregazione non viene eseguito.

Per il caso 2 si è scelto di modulare la soglia in modo lineare incrementando il proprio valore di 0.2 ogni unità d'incremento della deviazione standard a partire da 5. La Figura 5.9 riporta la scelta della soglia al variare della deviazione standard delle query ricevute.

Tuttavia la deviazione standard delle query ricevute da un nodo non è un'informazione nota a priori ma dev'essere calcolata. Questo può essere fatto in due maniere: una valutazione precisa attraverso un'analisi

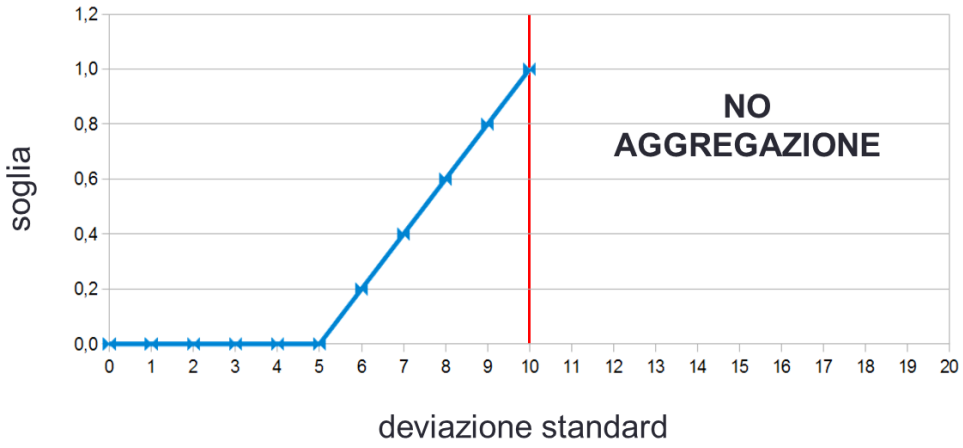

Figura 5.9: Selezione della soglia di aggregazione al variare della deviazione standard delle query.

approfondita delle query oppure una stima. Considerando che la valutazione precisa impiegherebbe ulteriormente il processore durante la fase di aggregazione, si è deciso quindi di stimare tale valore. In particolare, durante la fase di test si è trovata una relazione tra il numero di cluster generati dall'algoritmo di partizionamento e la deviazione standard delle query in ingresso. In particolare, la Figura 5.10 riporta tale relazione.

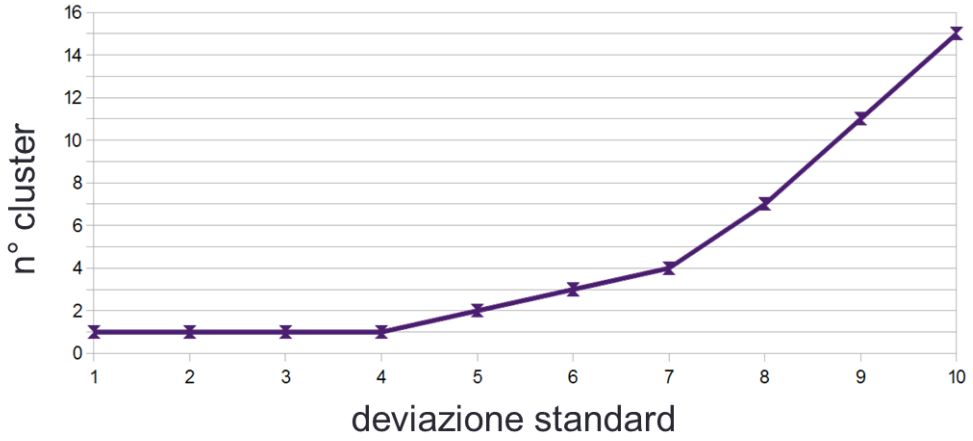

Figura 5.10: Relazione tra la deviazione standard delle query ricevute e il numero di cluster calcolati dall'algoritmo di partizionamento.

Quindi partendo dal numero di cluster calcolati dall'algoritmo di par-

tizionamento si stima la deviazione standard delle query in ingresso che servirà per modulare la soglia di aggregazione.

5.3.4 Modulazione del carico di lavoro del processore

In questa soluzione, proposta per dispositivi mobili, bisogna considerare anche il carico di lavoro del processore durante la fase di aggregazione. A causa delle scarse risorse computazionali il dispositivo potrebbe non essere sempre in grado di eseguire l'elaborazione dell'algoritmo. Si ipotizzi, a titolo di esempio, un telefono cellulare che durante un certo periodo di tempo effettui delle operazioni ad elevato dispendio di calcolo come la visualizzazione di un video o l'aggiornamento di un applicativo. In tali istanti l'utilizzo di CPU potrebbe essere molto elevato, tale da occupare tutte le risorse di sistema e non consentire l'elaborazione dell'algoritmo di aggregazione.

Per questo motivo, l'ultimo passo implementativo della soluzione per dispositivi mobili è stato quello di controllare il carico di lavoro del processore durante la fase di aggregazione. In particolare si monitora un indicatore di utilizzo del processore prima dell'esecuzione dell'algoritmo e, se il valore supera una soglia prefissata, l'algoritmo non viene eseguito. Inoltre, tale controllo viene effettuato ad ogni ciclo dell'algoritmo di aggregazione e, se il valore supera la soglia, l'algoritmo termina alla fine del ciclo. È stato creato un componente che monitora costantemente la percentuale di utilizzo di CPU ed informa il processo di aggregazione filtrando le variazioni troppo repentine mediante la seguente funzione:

$$mean\_cpu = (1 - alpha) \cdot mean\_cpu + alpha \cdot instant\_cpu.$$

con $alpha$ una costante fissata a 0.7.

5.4 Soluzione per dispositivi ad elevate capacità computazionali

Per i dispositivi dotati di elevata capacità computazionale, tipicamente i nodi fissi, il problema di riduzione del carico di lavoro del processore assume, nella maggior parte dei contesti, un'importanza secondaria. Considerando l'architettura distribuita di SALES in Figura 5.1, si è progettata una soluzione da eseguire sui nodi fissi (BN e CN) al fine di ridurre il numero di risposte inviate ai livelli sottostanti sfruttando sempre la sovrapposizione delle query.

Dai test effettuati utilizzando la soluzione precedente, si è notato che il traffico aumenta già per deviazioni standard non troppo elevate (tra 4 e 5). Quindi, seppur si riesce ad avere un risparmio di lavoro del processore, si stanno introducendo un numero elevato di dati spuri tale da aumentare il traffico dati generale. La Figura 5.11 mostra il confronto tra i traffici dati generati dai diversi test. Tali risposte sono state generate a partire dal confronto tra il risultato dell'aggregazione di 100 query, ognuna delle quali richiede da 4 a 8 dati, e l'intero insieme di dati che possono essere richiesti.

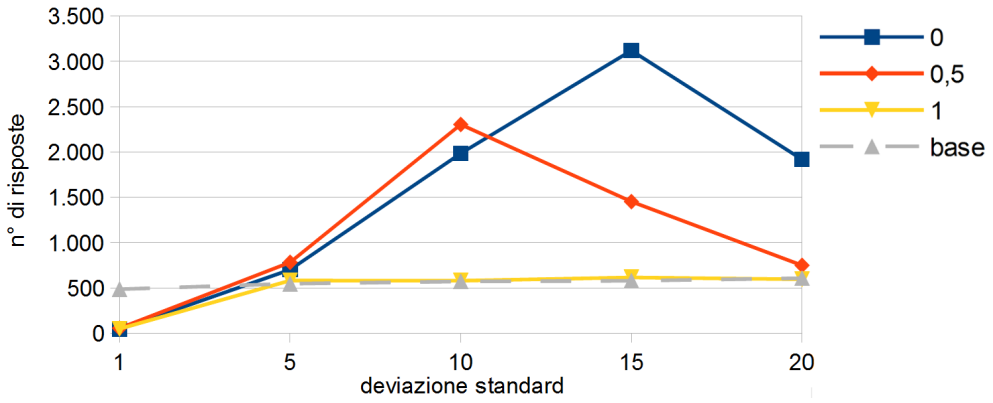

Figura 5.11: Traffico dati generato al variare della somiglianza delle query e della soglia di aggregazione utilizzando la soluzione proposta per dispositivi mobili.

In questo caso, a differenza della precedente soluzione, non si prende in considerazione il carico di lavoro del processore. Per tale motivo, la modulazione del carico di lavoro del processore, proposta nel Sottoparagrafo 5.3.2, ed il calcolo della soglia per modulare il processo si aggregazione, proposta nel Sottoparagrafo 5.3.3, non dovranno essere eseguiti.

La soluzione proposta per dispositivi fissi si articola quindi in sole due fasi:

1. partizionamento dell'insieme delle query ricevute
2. esecuzione dell'algoritmo di aggregazione per ogni cluster.

Siccome l'algoritmo di partizionamento è identico a quello realizzato per i dispositivi mobili, verrà illustrato solo il nuovo algoritmo di aggregazione.

5.4.1 Algoritmo di aggregazione

Per minimizzare il numero di risposte bisogna evitarne l'invio multiplo verso uno stesso nodo. Nella versione originale SALES prevede che un nodo invii una copia di un dato per ogni query che soddisfa, anche se il destinatario è lo stesso; questo è causato dal fatto di non considerare le similarità tra le query. Il comportamento che si vuole realizzare prevede, invece, che il dato venga inoltrato una volta sola per ogni nodo interessato.

Per la realizzazione si è deciso di sfruttare la rappresentazione logica delle query: l'algoritmo di aggregazione prevede che tutte le query simili, ovvero contenute all'interno dello stesso cluster, vengano unite in OR in un'unica espressione logica. Quindi l'output del processo di aggregazione prevede un'unica query ORFC di:

- tutte le query ANDFC
- tutte le query MTF aggregando i MTF con lo stesso attributo
- tutte le sotto-query delle query ORFC.

Un esempio grafico degli effetti di tale aggregazione viene proposto in Figura 5.12, dove le query q_1, q_2 e q_3 vengono aggregate nella query q_a.

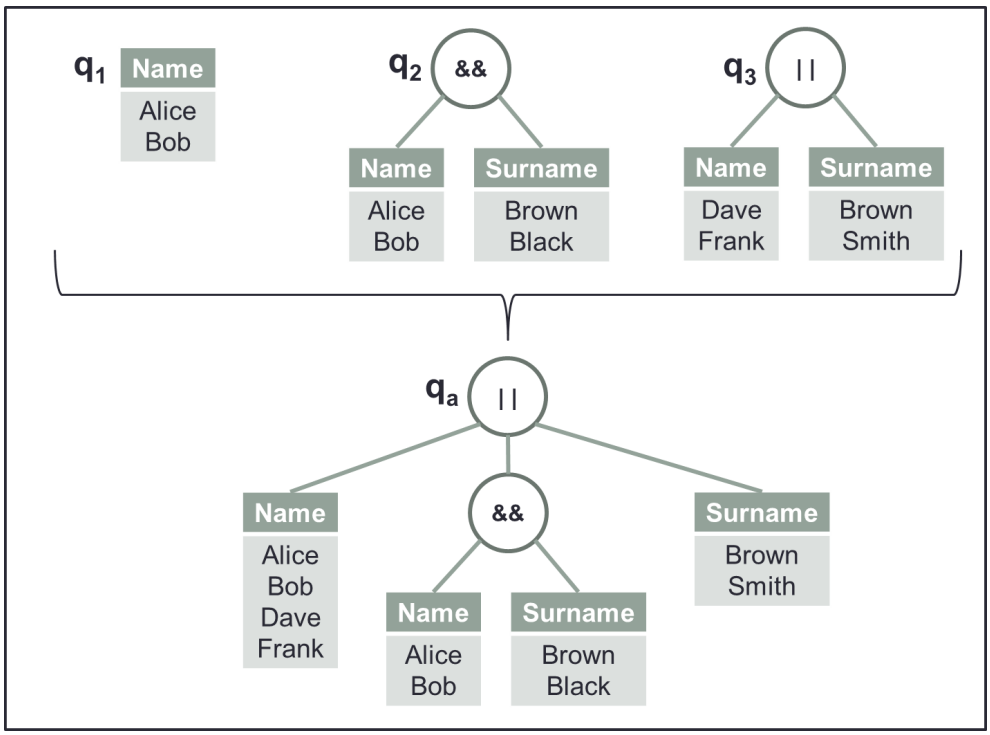

Figura 5.12: Aggregazione adatta all'esecuzione su dispositivi fissi.

Non prevedendo il rilassamento delle query tale algoritmo non introduce dati spuri. Sebbene il numero di query sarà ridotto notevolmente, si osservi che in fase di ricezione dei dati non è previsto un risparmio significativo di lavoro del processore poiché le query da elaborare risultano più complesse.

5.5 Riduzione dei picchi di utilizzo del processore

Come specificato nel Paragrafo 5.1, ogni volta che un nodo riceve un certo numero di query può decidere di eseguire l'algoritmo di aggregazione. Al fine di ridurre i picchi di utilizzo di CPU si è pensato di distribuire unifor-

memente nel tempo l'esecuzione dell'algoritmo. Per prima cosa bisogna stimare il tempo a disposizione per l'esecuzione e successivamente bisogna rallentare l'algoritmo.

Fissato il numero di query da ricevere prima di eseguire l'algoritmo, la stima sul tempo disponibile viene fatta in base al tempo trascorso per accumulare le query. Quindi si fa partire un timer alla ricezione della prima query e lo si interrompe alla ricezione dell'ultima, prima di effettuare l'aggregazione. A partire dal tempo calcolato si stima che il successivo insieme di query sarà ricevuto nello stesso arco temporale e quindi si stabilisce di avere a disposizione l'intero arco di tempo prima di dover eseguire nuovamente l'algoritmo di aggregazione sul nuovo insieme di query ricevute. Per fissare le idee si consideri di voler eseguire l'algoritmo di aggregazione ogni 100 query; se le query sono state ricevute in 10 secondi, allora si stima di ricevere le successive 100 in altri 10 secondi e quindi di avere 10 secondi a disposizione per l'esecuzione dell'algoritmo. In realtà, per far fronte a variazioni di frequenza di ricezione delle query, si considera di avere circa il 75% del tempo disponibile, ovvero 7.5 secondi nel precedente esempio.

Partendo dalla stima del tempo disponibile e dal numero di query considerate viene calcolato un tempo di ritardo da poter inserire all'interno di ogni ciclo dell'algoritmo, sia durante il partizionamento che durante l'aggregazione. Per ritardare l'esecuzione si è utilizzata la primitiva *static void sleep(long millis, int nanos)* della classe *Thread* di $Java$ che consente di fermare l'esecuzione del processo per un tempo minimo di *millis* millisecondi e *nanos* nanosecondi.

Capitolo 6

Risultati sperimentali

Per validare il lavoro di questa tesi sono stati effettuati diversi esperimenti su testbed reali. I risultati mettono in luce i miglioramenti delle performance introdotti dall'aggregazione.

Il Paragrafo 6.1 descrive i deployment utilizzati riportando le principali caratteristiche delle macchine. Nel Paragrafo 6.2 vengono descritti i modelli utilizzati per le query e come avviene la loro generazione; inoltre si descrivono l'insieme di dati prodotto e le loro caratteristiche. Nel Paragrafo 6.3 e nel Paragrafo 6.4 si riportano e si analizzano i risultati di alcuni degli esperimenti effettuati sfruttando, rispettivamente, la parte di soluzione pensata per dispositivi mobili e quella per dispositivi fissi. Nel Paragrafo 6.5 vengono descritti i risultati ottenuti dall'utilizzo congiunto di entrambe le parti della soluzione. Infine, nel Paragrafo 6.6 si riportano alcune considerazioni a conclusione del capitolo.

6.1 Deployment utilizzati

Lo scopo di questo paragrafo è quello di descrivere le principali caratteristiche HW/SW dei dispositivi utilizzati e di come essi siano stati connessi per effettuare gli esperimenti. Essendo la soluzione implementata in $J2SE$, non è stato possibile utilizzare i telefoni cellulari; l'utilizzo di soli dispositivi ad elevate prestazioni non limita la validità di questa trat-

tazione poiché si sono utilizzate delle soglie di modulazione del carico di lavoro molto basse.

Le macchine a disposizione, le relative caratteristiche ed i ruoli ad esse assegnati per i deployment sono riportati nella Tabella 6.1.

Tipo	Processore	RAM	S.O.	Ruolo
Desktop	2x Intel Core 2 Duo CPU E7400 @ 2.80GHz	1.9 GB	Linux Ubuntu 9.04 (juanti)	BN
Desktop	2x Intel Pentium Dual CPU E2160 @ 1.80GHz	2.0 GB	Linux Ubuntu 11.10 (oneiric)	CUN
Laptop	AMD Turion X2 Dual-Core Mobile RM-75 @ 2.20GHz	3.7 GB	Linux Ubuntu 12.04 (precise)	SUN_1
Laptop	2x Intel Pentium Dual CPU T2370 @ 1.73GHz	1.8 GB	Linux CentOS 6.3 (final)	SUN_2

Tabella 6.1: Caratteristiche dei dispositivi utilizzati.

Con tali dispositivi si sono realizzati due deployment, uno più semplice per i primi test e l'altro più articolato. Il primo, chiamato D_1, prevede la presenza del solo CUN che emette le query e BN che esegue l'algoritmo di aggregazione e genera i dati; il secondo, chiamato D_2, prevede la presenza di tutti 4 i nodi descritti nella Tabella 6.1 con i SUN_1 e SUN_2 che emettono le query, il CUN che esegue l'algoritmo di aggregazione per nodi mobili ed il BN che esegue l'algoritmo di aggregazione per nodi fissi e genera i dati.

Per la connessione il BN è collegato via cavo attraverso lo standard Ethernet ad un access point *Cisco Aironet 1100* al quale il CUN si collega wireless tramite standard $IEEE$ 802.11. Come detto nel Paragrafo 3.3, il CUN deve essere dotato di doppia interfaccia in quanto costituisce il pun-

to di accesso tra l'infrastruttura mobile e quella fissa. Quindi, mediante un secondo dispositivo di rete wireless, consente al SUN_1 e al SUN_2 di collegarsi ad esso tramite standard $IEEE$ 802.11.

I due deployment sono riportati in Figura 6.1.

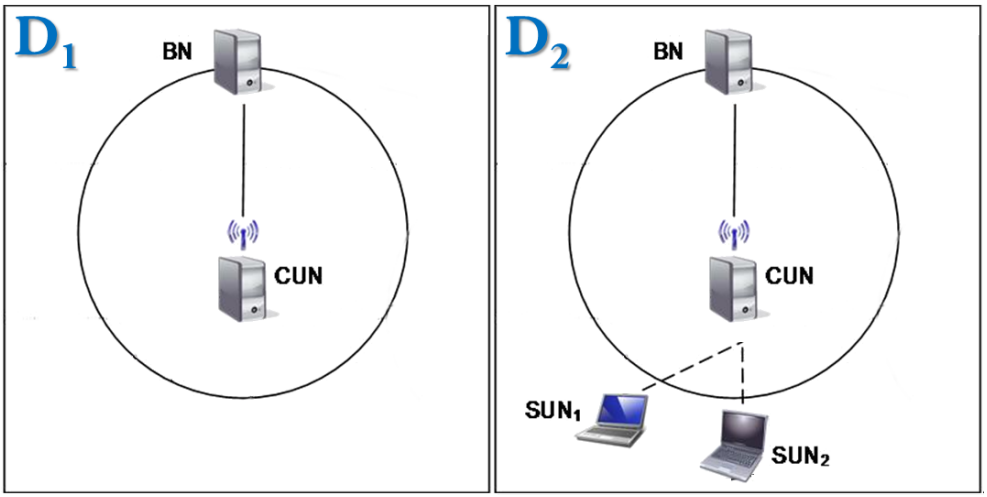

Figura 6.1: I deployment utilizzati durante gli esperimenti.

Si ipotizzano scenari in cui il CUN genera o riceve 10 query ogni secondo. Si fa presente, infine, che il nodo CUN è dotato di una cache in grado di contenere un massimo di 50 dati; tale scelta è coerente con le capacità dei dispositivi mobili. Si è deciso di considerare la cache durante gli esperimenti in quanto caratteristica integrante della versione originale di SALES.

6.2 Query e dati generati

Il tipo di dato utilizzato è stato descritto nel Paragrafo 3.5 ed è formato da due coppie attributo-valore ed un *payload*. Le query consentono di esprimere gli interessi verso i dati attraverso la specifica di possibili valori per gli attributi. Per tale motivo, si focalizza l'attenzione solo su questi ultimi

ignorando di fatto il *payload*.

Negli esperimenti effettuati gli attributi dei dati sono due: *identifier* e *value*; i possibili valori per il primo attributo sono $id_i, \forall i = 0, ..., 100$, mentre per il secondo sono $val_i, \forall i = 0, ..., 100$. L'intero insieme dei dati generati è formato dalla combinazione di tutti i possibili valori dei due attributi, quindi è composto da $101 \cdot 101 = 10201$ elementi.

Sono stati utilizzati due modelli di query con attributi fissi e valori variabili:

1. ANDFC tra due MTF, uno per specificare due possibili valori dell'attributo *identifier* e l'altro dell'attributo *value*

2. ORFC tra due ANDFC costruiti come al punto 1.

La Figura 6.2 riporta i due modelli di query e le proprietà che devono soddisfare.

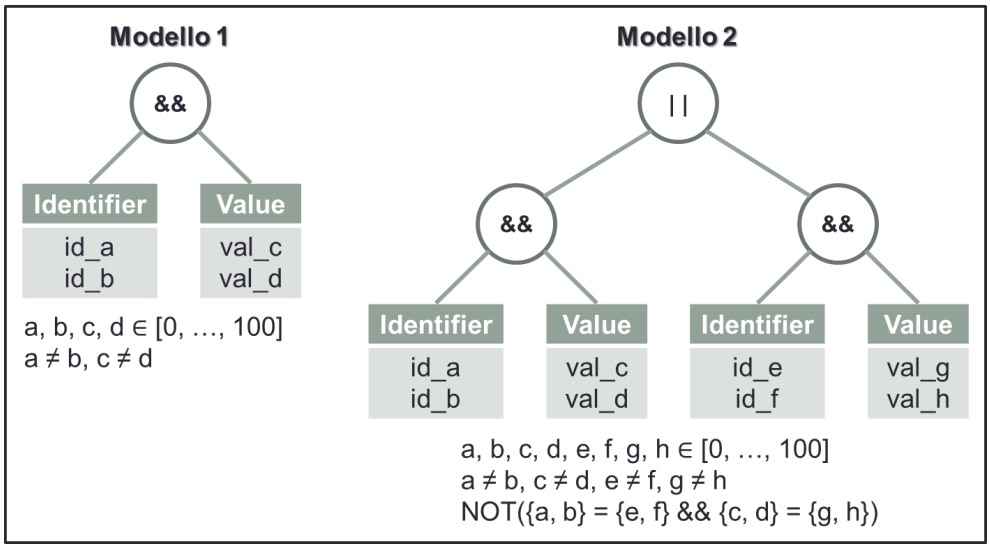

Figura 6.2: I due modelli di query utilizzati per gli esperimenti.

La scelta dei modelli di query da utilizzare durante gli esperimenti è stata fatta partendo da alcune considerazioni. Anzitutto, non si ritiene opportuno utilizzare query formate da un unico MTF in quanto, specificando

un solo attributo, richiederebbero un numero elevato di dati ed avrebbero quindi un'elevata probabilità di sovrapposizione con molte query, anche in copertura totale. Poi, si vogliono utilizzare almeno due modelli di query: uno semplice ed uno più complesso. Infine, come riportato nel Sottoparagrafo 5.3.2, le espressioni booleane possono essere rappresentate mediante somma di prodotti o prodotti di somma; in questo caso, per la query complessa, si è scelta la forma somma di prodotti.

Ogni query del primo modello richiede sempre 4 dati mentre le query del secondo modello possono richiedere da 6 dati, quando le due sotto-query hanno 3 valori in comune e quindi richiedono 2 dati uguali, a 8 dati, quando le due sotto-query non richiedono dati in comune, escludendo il caso in cui le due sotto-query sono identiche. Per ogni query da generare si sceglie uno dei due modelli in maniera pseudo-random mentre i valori degli attributi vengono scelti secondo una distribuzione gaussiana di valor medio pari a 50 e deviazione standard scelta in base all'esperimento. È possibile stimare il numero di dati richiesti con la seguente formula:

$$stima\ dati\ richiesti = \frac{|Q|}{2} \cdot 4 + \frac{|Q|}{2} \cdot \frac{6+8}{2} = \frac{11}{2} \cdot |Q| \,.$$

6.3 Soluzione per dispositivi a risorse scarse

La soluzione realizzata per dispositivi a risorse scarse, descritta nel Paragrafo 5.3, è stata testata dapprima in maniera isolata con il deployment D_1. Per tutti i test si è deciso di eseguire l'algoritmo di aggregazione ogni 100 query ricevute allo scopo di avere una dimensione del problema non troppo piccola, per non rendere poco efficace l'aggregazione, né troppo grande, per non rendere difficoltosa l'esecuzione. Per monitorare l'utilizzo di CPU nel tempo durante più esecuzioni dell'algoritmo di aggregazione, sono state prodotte 1000 query per ogni esperimento; quindi l'esecuzione dell'algoritmo di aggregazione viene scatenata 10 volte.

Con il deployment D_1 sono stati effettuati diversi esperimenti di aggregazione per i seguenti casi:

- sfruttando tutta la capacità di calcolo
- limitando l'uso del processore al 15%
- limitando l'uso del processore al 10%

Per verificare il miglioramento introdotto è stata considerata anche la versione originale di SALES. Tutti i casi sono stati valutati per diversi valori di deviazione standard delle query prodotte: 1.0, 5.0, 7.5, 10.0 e 12.5. I primi esperimenti sono stati eseguiti producendo l'intero insieme di dati con una frequenza pari ad un dato ogni 100 millisecondi, in linea con i parametri delle classi di test già presenti in SALES; questo ha causato la perdita di associazione tra il BN ed il CUN in quanto quest'ultimo non in grado di gestire l'elevata frequenza dei messaggi ricevuti. Allora si è pensato di generare i dati con una frequenza più bassa, ma questo ha reso difficile il rilevamento del vantaggio introdotto dalla soluzione. È per questo motivo che si è deciso, infine, di generare solamente i dati statisticamente più richiesti con una frequenza di un dato ogni 100 millisecondi. Il numero di dati generati è 25 e sono tutti quelli compresi tra (id_{48}, val_{48}) e (id_{52}, val_{52}).

La Figura 6.3 riporta a confronto i risultati di vari esperimenti con deviazione standard pari a 5.0. Ogni esperimento è stato graficamente diviso in due parti: a sinistra si riporta l'utilizzo del processore durante un intervallo di tempo limitato della fase di aggregazione (circa 40 secondi); mentre a destra si riporta tutta la fase di interrogazione. In linea continua si rappresentano gli esperimenti che sfruttano l'aggregazione; nello specifico: in alto si ha il caso in cui l'algoritmo di aggregazione non ha limiti nell'utilizzo della CPU, al centro il caso in cui si limita l'utilizzo di CPU di sistema al 15% ed in basso il caso in cui il limite è fissato al 10%. In linea tratteggiata, invece, si ha il caso originale, ovvero senza aggregazione. I valori riportati sono stati prelevati dal nodo BN.

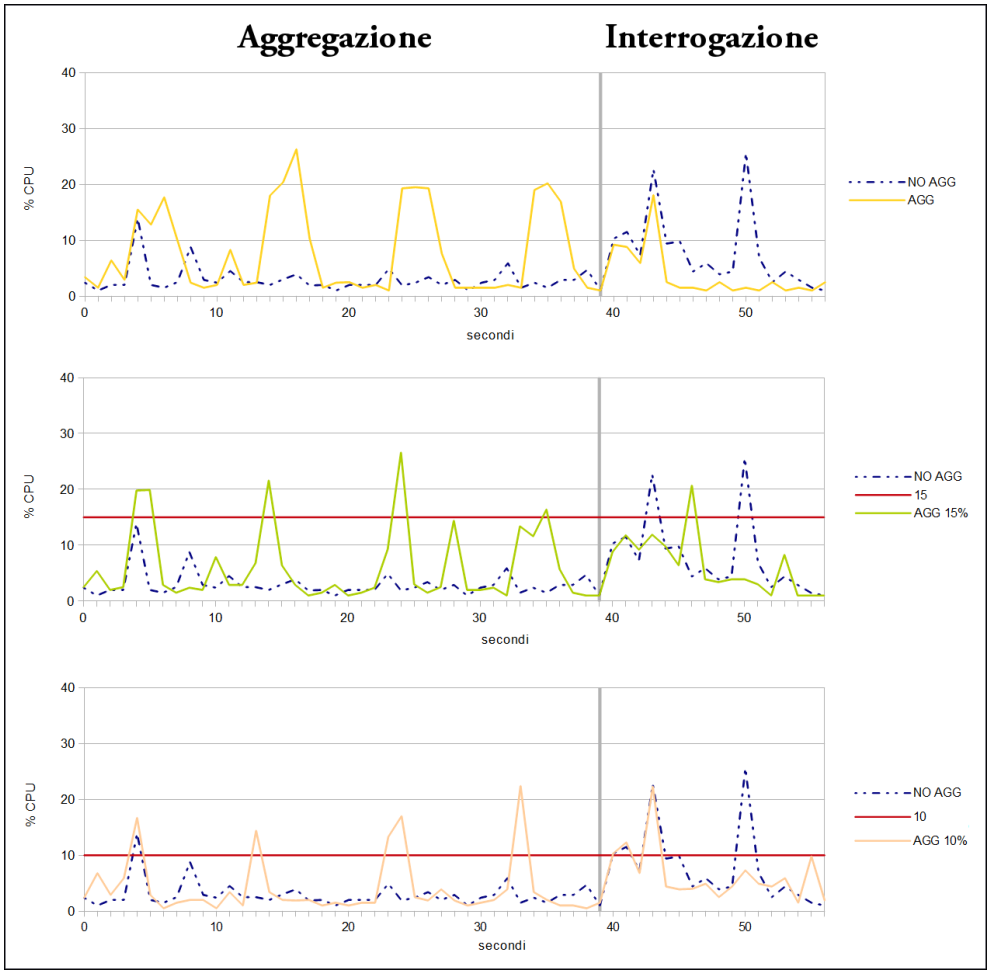

Figura 6.3: Utilizzo di CPU per deviazione standard pari a 5.0.

Dai grafici si possono effettuare le seguenti osservazioni:

- L'esperimento che utilizza la versione originale di SALES viene ripetuto su tutti i grafici in linea tratteggiata. Nella fase di aggregazione non si ha un elevato carico computazionale poiché si ricevono le query senza eseguire l'algoritmo di aggregazione; nella fase di interrogazione, invece, la versione originale di SALES ha un carico computazionale complessivo maggiore rispetto alla soluzione proposta.
- L'esperimento che sfrutta l'aggregazione senza limiti di utilizzo di CPU viene presentato nel grafico in alto in linea continua. Durante la

fase di aggregazione si notano 4 *archi* che rappresentano 4 esecuzioni dell'algoritmo; nella fase di interrogazione si nota chiaramente che l'utilizzo di CPU si è ridotto notevolmente rispetto al caso originale.

- L'esperimento che sfrutta l'aggregazione limitando l'utilizzo di CPU di sistema al 15% viene proposto nel grafico centrale in linea continua. Durante la fase di aggregazione le rappresentazioni delle esecuzioni dell'algoritmo si trasformano in picchi di carico di durata più limitata poiché l'algoritmo viene interrotto al superamento della soglia fissata; a causa di tali interruzioni, in fase di interrogazione, il risparmio nell'utilizzo di CPU si riduce.

- L'esperimento che sfrutta l'aggregazione limitando l'utilizzo di CPU di sistema al 10% viene presentato nel grafico in basso in linea continua. Valgono le stesse considerazioni del caso precedente, considerando che una limitazione maggiore in fase di aggregazione riduce ulteriormente il risparmio in fase di interrogazione.

Dall'analisi numerica dei risultati si riscontrano i risparmi, in termini di $\%$ di CPU utilizzata in fase di interrogazione, mostrati nella Tabella 6.2.

	% Risparmio CPU		
Algoritmo	Aggregazione	Aggregazione 15%	Aggregazione 10%
Risparmio	50%	20%	19%

Tabella 6.2: Risparmi di CPU ottenuti tramite la soluzione per dispositivi mobili per deviazione standard pari a 5.

Inoltre, si vogliono mostrare i diversi andamenti di utilizzo di CPU anche per diverse deviazioni standard; si è scelto di mostrare i risultati degli esperimenti che sfruttano l'aggregazione limitando l'utilizzo di CPU di sistema al 15% in quanto ritenuto il più significativo. A causa della limitatezza dell'insieme dei dati prodotto, all'aumentare della deviazione

standard ci si aspetta che ci siano meno query soddisfatte e quindi meno dati in risposta. La Figura 6.4 riporta i grafici dei test eseguiti; anche in questo caso i valori sono stati estratti dal nodo BN.

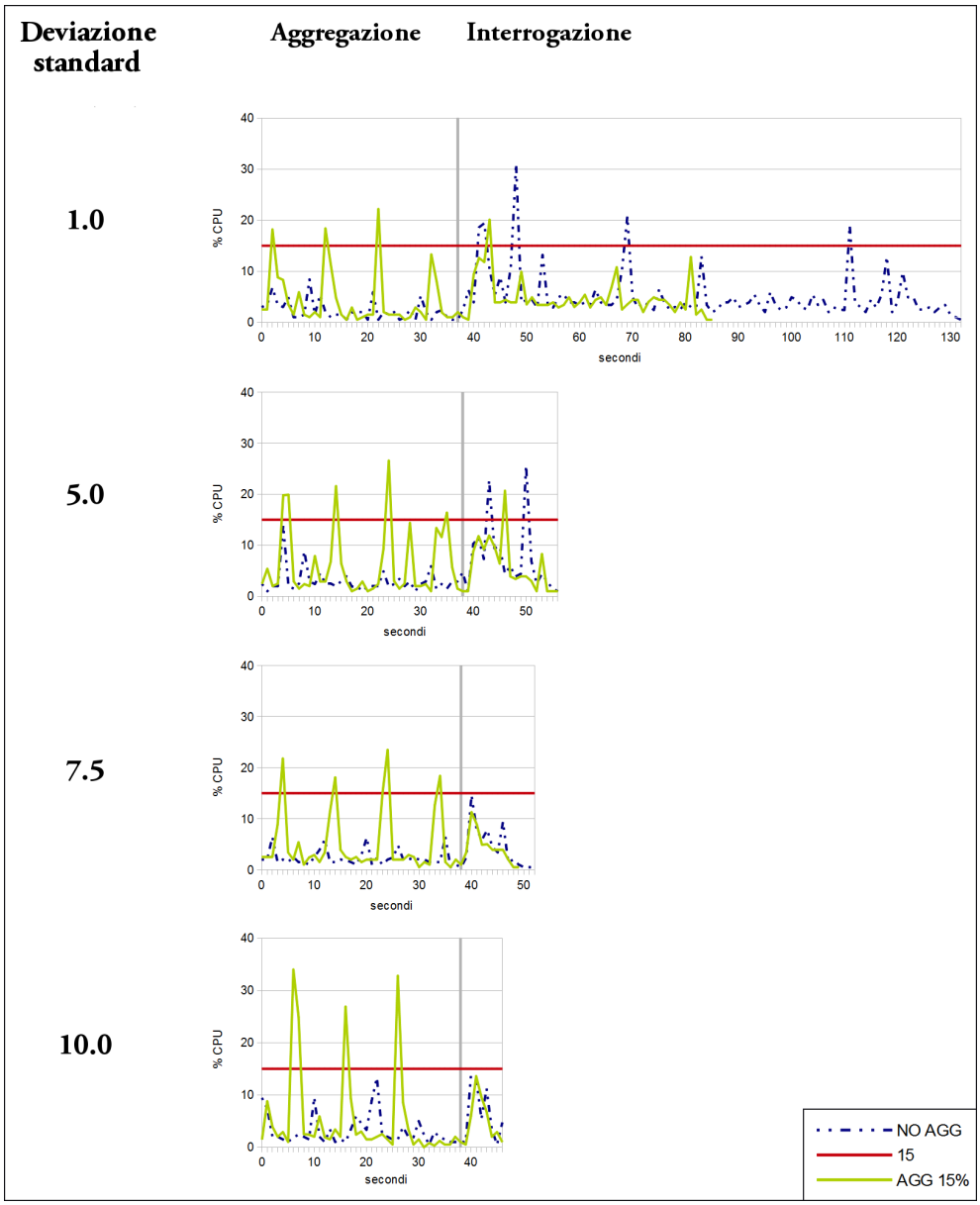

Figura 6.4: Utilizzo di CPU con limite aggregazione al 15%.

Come per i grafici precedenti, in linea tratteggiata si riporta il caso senza aggregazione mentre in linea continua il caso in cui si sfrutta l'aggregazione. Nella versione originale le query non vengono aggregate perciò, al variare della deviazione standard, non si osservano differenze di comportamento significative nella prima fase; nella fase di interrogazione, come spiegato prima, il numero di dati da inviare si riduce all'aumentare della deviazione standard ed è per questo che il carico computazionale ed il tempo impiegato per inviare i dati diminuisce.
Nella versione che sfrutta l'aggregazione, invece, l'utilizzo di CPU tenderebbe a diminuire perché si aggrega sempre meno a causa dell'aumento dinamico della soglia di aggregazione. Tale tendenza però viene contrastata dal fatto che, con l'aggregare delle query, gli insiemi dei valori ammissibili per gli attributi diventano sempre più grandi poiché le query sono sempre meno sovrapposte; quindi, l'inserimento di nuovi valori diventa più costoso dal punto di vista computazionale in quanto bisogna verificare se l'attributo da inserire sia già presente. Per deviazioni standard molto vicine al limite di 10.0, a causa della scarsa similarità delle query, l'algoritmo di aggregazione termina velocemente poiché vi sono pochissime coppie di query con un guadagno di banda intorno a 1.0. Nella fase di interrogazione, il risparmio di CPU diminuisce all'aumentare delle deviazione standard sia perché le query sono meno simili e sia perché si riduce il numero di dati da inviare.

Dall'analisi numerica dei risultati si riscontrano i risparmi, in termini di $\%$ di CPU utilizzata in fase di interrogazione, mostrati in Tabella 6.3.

	% Risparmio CPU			
Deviazione standard	1.0	5.0	7.5	10.0
Risparmio ($\%$)	53%	20%	18%	15%

Tabella 6.3: Risparmi di CPU ottenuti tramite la soluzione limitando il consumo di CPU al 15% in fase di aggregazione.

L'obiettivo principale della soluzione proposta per dispositivi mobili

è quello di ridurre il carico computazionale sul nodo che la esegue. Ciononostante, l'aggregazione condiziona anche il traffico dati generato dalle risposte; per questo motivo si riportano, per completezza, tali informazioni nella Tabella 6.4, in riferimento agli esperimenti mostrati in Figura 6.3, e nella Tabella 6.5, in riferimento agli esperimenti mostrati in Figura 6.4.

Versione SALES	**Risposte inviate**	**Risparmio (%)**
Originale	858	N.A.
Aggregazione 10%	630	27%
Aggregazione 15%	618	28%
Aggregazione	252	71%

Tabella 6.4: Informazioni sul traffico dati delle risposte per deviazione standard pari a 5.

	Risposte inviate - Risparmio (%)	
Deviazione standard	**Originale**	**Aggregazione** 15%
1.0	5359	1382 - 74%
5.0	858	618 - 28%
7.5	375	280 - 25%
10.0	223	176 - 21%

Tabella 6.5: Informazioni sul traffico dati delle risposte generato dall'algoritmo di aggregazione limitando l'utilizzo di CPU di sistema al 15%.

È opportuno ricordare che sono stati generati solamente i dati statisticamente più richiesti, quindi non sono stati riscontrati aumenti di banda neppure per deviazioni standard elevate. Tuttavia, si pensa che generando dati distanti dalla media, il traffico dati potrebbe aumentare rispetto al caso originale.

Infine, è stato effettuato un esperimento per valutare l'utilizzo di CPU durante la fase di aggregazione, variando dinamicamente la deviazione standard delle query in ingresso. Il nodo CUN genera le query impo-

stando una differente deviazione standard per ogni gruppo di 100 query inviate; il primo gruppo viene generato con una deviazione standard pari a 3.0, i successivi subiranno un incremento di una unità fino ad una deviazione standard pari a 12.0. Il nodo BN riceve le query e, per ogni 100 ricezioni, esegue l'algoritmo di aggregazione senza limiti sull'utilizzo delle risorse di calcolo. La Figura 6.5 mostra la percentuale di CPU utilizzata dal sistema durante la fase di aggregazione, la deviazione standard delle query ricevute e quella stimata dall'algoritmo.

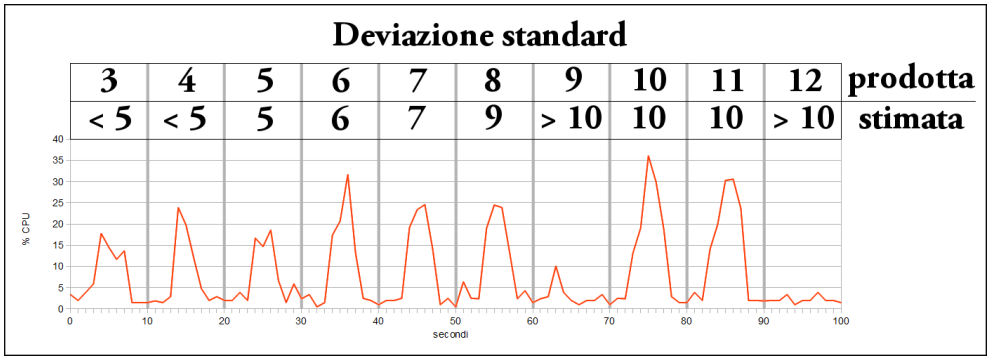

Figura 6.5: Utilizzo di CPU durante la fase di aggregazione al variare dinamico della similarità delle query sfruttando l'algoritmo senza limiti di utilizzo delle capacità di calcolo.

Come mostrato precedentemente nella Figura 5.10, per deviazioni standard minori di 5 non è possibile effettuare una stima più accurata in quanto il numero di cluster calcolati è sempre pari a 1. Si ricorda, inoltre, che tale informazione non è stata ritenuta indispensabile per adattare la soglia di aggregazione. Per deviazioni standard maggiori di 10, l'algoritmo non viene eseguito e perciò non è necessario avere una stima più dettagliata. Il range ritenuto interessante per adattare il processo di aggregazione comprende deviazioni standard da 5 a 10; in questo esperimento, la stima risulta spesso combaciare con la deviazione standard prodotta o, al limite, non si discosta di una quantità tale da rendere inopportuna la scelta del metodo di stima. Per l'analisi sull'utilizzo di CPU al variare della deviazione standard, valgono le stesse considerazioni esposte precedentemente per la Figura 6.4.

6.4 Soluzione per dispositivi ad elevate capacità computazionali

La soluzione realizzata per dispositivi ad elevate capacità computazionali, descritta nel Paragrafo 5.4, è stata testata inizialmente con il deployment D_1. In questo caso, a differenza della soluzione per dispositivi mobili, si è deciso di eseguire l'algoritmo di aggregazione sul nodo BN dopo aver ricevuto 600 query in quanto l'elaborazione è molto più leggera e, in aggiunta, sui dispositivi fissi non si fanno ipotesi limitative. Dalla formula proposta precedentemente, è possibile stimare il numero di dati richiesti in circa 3300.

Sono stati effettuati diversi esperimenti di aggregazione al variare della deviazione standard delle query prodotte per i valori 1.0, 5.0, 10.0, 15.0 e 20.0. Per valutare l'effettivo risparmio di banda, è stato generato l'intero insieme di dati considerati: da (id_0, val_0) a (id_{100}, val_{100}).

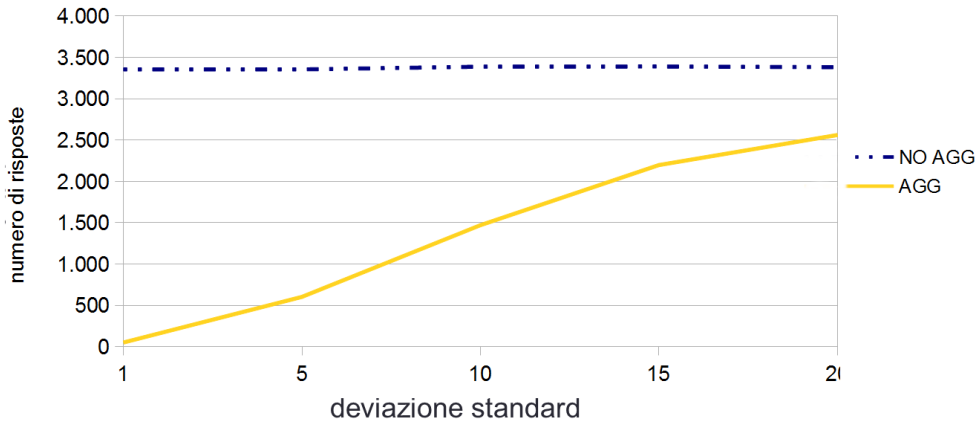

Figura 6.6: Traffico di risposte: confronto tra la versione originale di SALES e quella proposta in questa tesi.

La Figura 6.6 mostra il traffico di risposte in termini di numero di dati inviati dal nodo BN al nodo CUN generato dalla versione originale di SALES, linea tratteggiata, e quello generato dalla soluzione proposta, linea continua. Nei casi in cui le query sono molto simili il risparmio di banda

è enorme, ma si osserva un notevole risultato anche quando le query sono meno simili. Le percentuali di risparmio di traffico sono mostrati nella Tabella 6.6.

	% Risparmio traffico risposte				
Deviazione standard	1.0	5.0	10.0	15	20.0
Risparmio	98%	82%	57%	35%	24%

Tabella 6.6: Risparmi del traffico delle risposte.

Sono stati eseguiti gli stessi esperimenti effettuati per la soluzione proposta per dispositivi mobili e, per le stesse motivazioni, è stato generato un insieme limitato di dati: da (id_{48}, val_{48}) a (id_{52}, val_{52}). Si è monitorato l'utilizzo di CPU sul nodo CUN al fine di quantificare il risparmio di risorse sul nodo mobile.

La Figura 6.7 mette a confronto, per diverse deviazioni standard, l'utilizzo di CPU durante la fase di interrogazione sia per il caso originale che per il caso di aggregazione. Valgono le stesse osservazioni fatte nel Paragrafo 6.3 riguarda la fase di interrogazione per quanto concerne la produzione di dati; inoltre si osserva che al diminuire della similarità tra le query, diminuisce il vantaggio introdotto dalla soluzione.

Dall'analisi numerica dei risultati si riscontrano i risparmi mostrati in Tabella 6.7, in termini di $\%$ di CPU utilizzata in fase di interrogazione sul nodo CUN.

	% Risparmio CPU		
Deviazione standard	5.0	7.5	10.0
Risparmio	45%	25%	1%

Tabella 6.7: Risparmi di CPU ottenuti sul nodo CUN sfruttando l'aggregazione per nodi fissi.

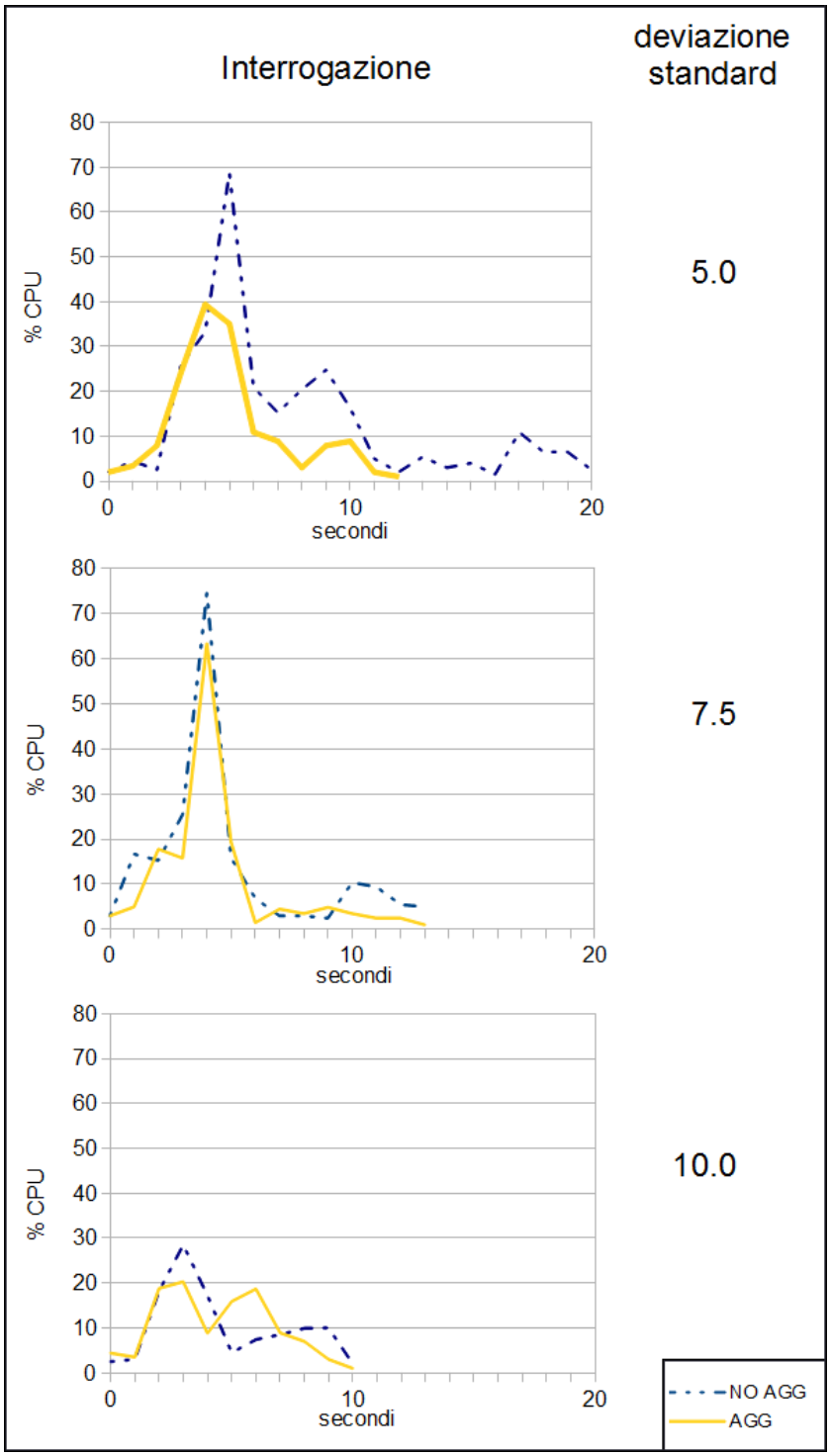

Figura 6.7: Utilizzo di CPU sul nodo CUN sfruttando l'aggregazione per nodi fissi.

6.5 Utilizzo congiunto delle due soluzioni

Infine si è deciso di testare l'utilizzo congiunto di entrambe le soluzioni: quella per dispositivi fissi e quella per dispositivi mobili. In particolare, utilizzando il deployment D_2, mostrato in Figura 6.1, il nodo BN esegue l'algoritmo di aggregazione realizzato per dispositivi ad elevate capacità computazionali mentre il nodo CUN esegue l'algoritmo realizzato per dispositivi a risorse scarse. I nodi SUN_1 e SUN_2 generano 300 query ognuno con una frequenza di 5 query al secondo, il nodo CUN elabora l'algoritmo di aggregazione ogni 100 query ricevute e il nodo BN esegue l'algoritmo di aggregazione al termine della ricezione di tutte le 600 query.

L'utilizzo di CPU sul nodo CUN in fase di interrogazione viene mostrato in Figura 6.8. In linea tratteggiata si riportano gli esperimenti che utilizzano la versione originale di SALES. In linea continua sottile si riporta la versione che sfrutta l'aggregazione utilizzando liberamente il processore. Infine, in linea continua larga si riposta la versione che sfrutta l'aggregazione limitando l'uso di CPU di sistema al 15%. Le deviazioni standard considerate sono 5.0, 7.5 e 10.0.

Dai grafici, in linea con gli esperimenti eseguiti precedentemente, si osserva un risparmio nell'utilizzo di CPU che decresce all'aumentare della deviazione standard; inoltre, limitando l'utilizzo di CPU in fase di aggregazione, si riduce il risparmio in fase di interrogazione.

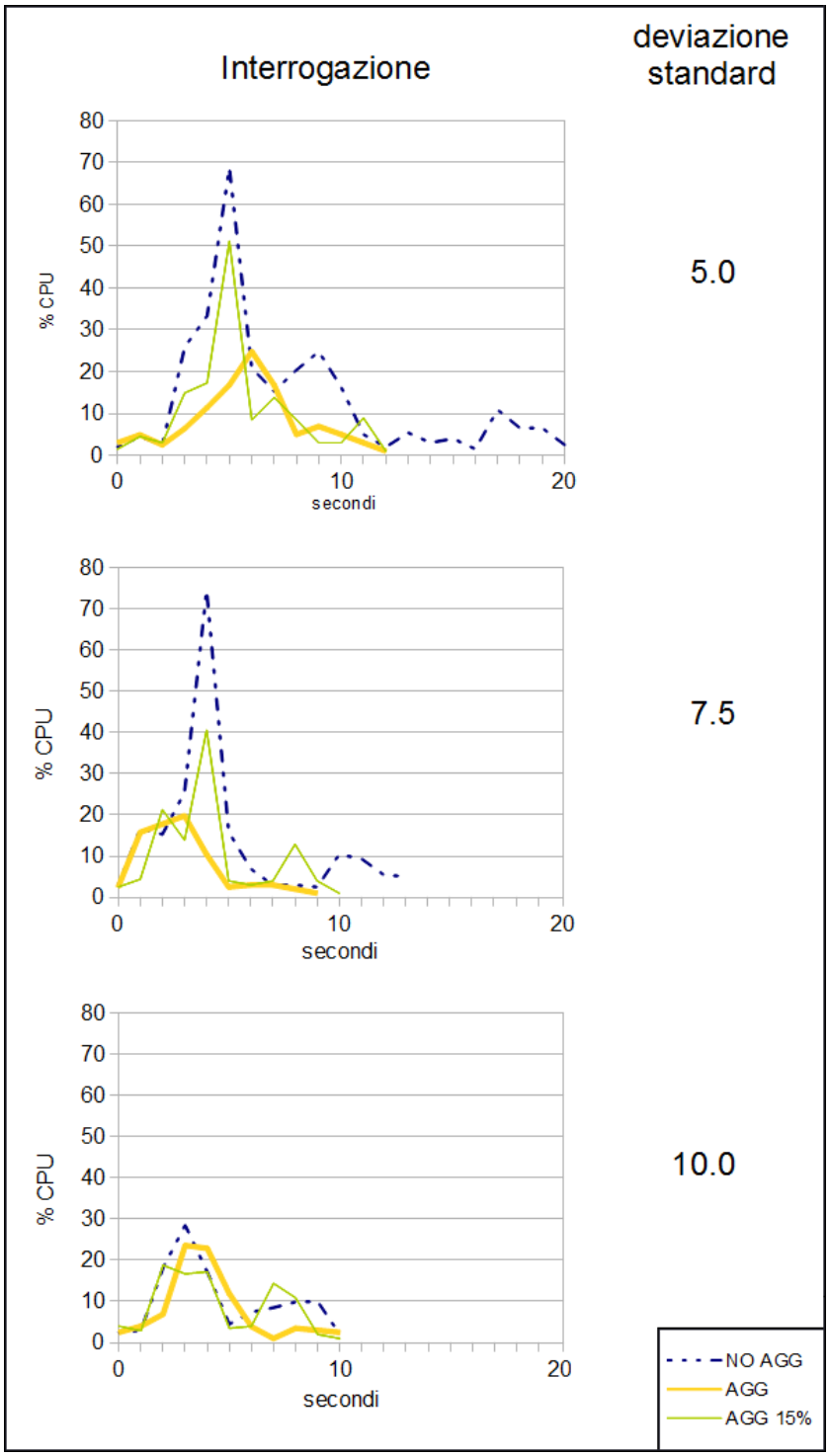

Figura 6.8: Utilizzo di CPU sul nodo CUN ottenuto sfruttando entrambe le parti della soluzione proposta.

Dall'analisi numerica dei risultati si riscontrano i risparmi mostrati in Tabella 6.8, in termini di $\%$ di CPU utilizzata in fase di interrogazione sul nodo CUN.

	% Risparmio CPU		
Deviazione standard	**5.0**	**7.5**	**10.0**
Risparmio aggregazione	65%	60%	23%
Risparmio aggregazione 15%	53%	43%	18%

Tabella 6.8: Risparmi di CPU ottenuti sul nodo CUN sfruttando l'aggregazione per dispositivi fissi sul nodo BN e l'aggregazione per dispositivi mobili sul nodo CUN. Nella penultima riga, il CUN esegue l'aggregazione senza limiti di utilizzo di CPU mentre, nell'ultima riga, esegue l'aggregazione limitando l'utilizzo di CPU di sistema al 15%.

6.6 Conclusioni

Analizzando i risultati di tutti gli esperimenti si può concludere che la soluzione proposta in questa tesi permette di ridurre il carico computazionale durante la fase di interrogazione, consentendo quindi di gestire una frequenza di ricezione dati più elevata. Il miglioramento delle performance dipende da diversi fattori: similarità tra le query, capacità computazionale disponibile in fase di aggregazione e quantità di query considerate. Inoltre, anche se le query non sono molto simili, si è dimostrato che con tale soluzione si riesce comunque ad ottenere un notevole risparmio del traffico dati dall'infrastruttura fissa verso quella mobile, riducendo drasticamente l'inoltro di dati duplicati.

Conclusioni

Il lavoro di tesi presentato si colloca nell'ambito dei sistemi context-aware basati sul modello publish/subscribe. Sono state indagate le principali tecniche di aggregazione delle sottoscrizioni allo scopo di ridurre il carico computazionale ed il traffico delle risposte. In particolare, è stata realizzata una soluzione sull'infrastruttura SALES in grado di ridurre il carico computazionale dei nodi mobili e il traffico delle risposte generato dall'infrastruttura fissa verso l'infrastruttura mobile.

La soluzione proposta è in grado di adattare il proprio comportamento in base alle risorse di calcolo disponibili sul dispositivo che la implementa. Per dispositivi dotati di scarse risorse computazionali, il principale obiettivo è la riduzione del carico di lavoro durante il confronto informazioni/sottoscrizioni. Per i dispositivi dotati di elevata capacità di calcolo si focalizza maggiormente l'attenzione sulla riduzione dell'invio di risposte duplicate verso i nodi mobili, così da ottenere un duplice vantaggio: il risparmio della banda ed un'ulteriore riduzione di carico computazionale sui nodi mobili.

I risultati ottenuti dagli esperimenti effettuati dimostrano che, al variare della similarità delle sottoscrizioni, si possono ottenere risparmi di computazione notevoli; inoltre, anche in caso di una ridotta similarità, si riesce comunque ad ottenere un risparmio di banda significativo. Si ritiene quindi che la soluzione introdotta apporti dei miglioramenti significativi di performance rispetto alla versione originale di SALES. Considerata la continua evoluzione delle tecnologie per dispositivi mobili, il sistema pro-

gettato si presta particolarmente a future integrazioni e miglioramenti. In tal senso, un possibile sviluppo potrebbe essere costituito dalla realizzazione di una versione adatta alla piattaforma mobile Android. Inoltre, si ritiene opportuno perfezionare la soluzione consentendo di adattare dinamicamente il numero di sottoscrizioni da aggregare in base alle capacità computazionali disponibili sul dispositivo. Infine, in situazioni di scarso utilizzo di risorse computazionali, si potrebbe trarre vantaggio dal considerare nuovamente l'aggregazione delle sottoscrizioni memorizzate in precedenza.

Bibliografia

[**Bac00**] Bacon J., Moody K., Bates J., R. Hayton Ma C., McNeil A., Seidel O. e Spiteri M.: *Generic support for distributed applications*. IEEE Comput. 33, 3 (Mar.), 68-76, 2000.

[**Ban99**] Banavar G., Chandra T., Mukherjee B., Nagarajarao J., Strom R e Sturman D.: *An efficient multicast protocol for content-based publish-subscribe systems*. In proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99), 1999a.

[**Ban02**] Banerjee S., Bhattacharjee B. e Kommareddy C.: *Scalable application layer multicast*. In proceedings of ACM SIGCOMM. ACM Press, New York, NY, 2002.

[**Bel12**] Bellavista P., Corradi A., Fanelli M. e Foschini L.: *A survey of context data distribution for mobile ubiquitous systems*. ACM Comput. Surv. 44, 4, Article 24 (August 2012), 45 pages, 2012.

[**Bir90**] Birman K., Cooper R., Joseph T., Marzullo K., Makpangou M., Kane K., Schmuck F. e Wood M.: *The Isis System Manual*. Dept. of Computer Science, Cornell University, Ithaca, NY, 1990.

[**Bir93**] Birman K.: *The process group approach to reliable distributed computing*. Commun., ACM 36, 12 (Dec.), 36-53, 1993.

[**Car00**] Carzaniga A., Rosenblum D. e Wolf A.: *Achieving scalability and expressiveness in a Internet-scale event notification service*. In proceedings of the Nineteenth ACM Symposium on Principles of Distributed Computing (PODC'00), ACM Press, New York, NY, 2000.

[**Cas02**] Castro M., Druschel P., Kermarrec A.-M. e Rowstron A.: *SCRIBE: A large-scale and decentralized application-level multicast infrastructure*. IEEE J. Sel. Areas Commun. 20, 8 (Oct.), 1489-1499, 2002.

[**Cha02**] Chan C.-Y., Fan W., Felber P., Garofalakis M. e Rastogi R.: *Tree pattern aggregation for scalable XML data dissemination*. In proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02), Hong Kong, China, 2002a.

[**Che00**] Chen G. e Kotz D.: *A survey of context-aware mobile computing research*. Techn. rep. TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[**Cor09**] Corradi A., Fanelli M. e Foschini L.: *Implementing a scalable context-aware middleware*. In Proceedings of the 14th IEEE International Symposium on Computers and Communications (ISCC'09), IEEE Computer Society Press, Sousse, Tunisia, July 5-8, 2009.

[**Cou12**] Couloris G., Dollimore J., Kindberg T. e Blair G.: *Distributed systems - Concepts and design (Fifth Edition)*. Addison-Wesley, 2012.

[**Cre03**] Crespo A., Buyukkokten O. e Garcia-Molina E.: *Query merging: improving query subscription processing in a multicast environment*. IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 1, January/February 2003.

[**Cug01**] Cugola G., Nitto E. D. e Fugetta A.: *The Jedi event-based infrastructure and its application to the development of the opss wfms*. IEEE Trans. Softw. Eng. 27, 9 (Sept.), 827-850, 2001.

[**Dee**] Deering S.: *Host extension for ip multicast*. IETF RFC 1112, Internet Engineering Task Force (Web site: www.ietf.org), N.D..

[**Dey99**] Dey A.K., Abowd G. D.: *Towards a better understanding of context and context-awareness*. Atlanta, 1999.

[**Eug00**] Eugster P., Guerraoui R. e Sventek J.: *Distributed asynchronous collections: abstractions for publish/subscribe interaction*. In Proceedings

of the 14th European Conference on Object-Oriented Programming (ECOOP'2000), 2000.

[**Eug01**] Eugster P. e Guerraoui R.: *Content-based publish/subscribe with structural reflection*. In Proceedings of the 6th Usenix Conference on Object-Oriented Technologies and Systems (COOTS'01), 2001.

[**Eug03**] Eugster P. T., Felber P. A., Guerraoui R. e Kermarrec A.-M.: *The Many Faces of Publish/Subscribe*. ACM Computing Surveys, Vol. 35, No. 2, pp. 114-131, June 2003.

[**Flo97**] Floyd S., Jacobson V., Liu C., McCanne S. e Zhang L.: *A reliable multicast framework for light-weight sessions and application level framing*. IEEE/ACM Trans. Netw. 5, 6, 784-803, 1997.

[**Fre99**] Freeman E., Hupfer S. e Arnold K.: *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley, Reading, MA, 1999.

[**Gra97**] Graham-Cumming J.: *Hits and miss-es: a year watching the web*. Sixth Int'l World Wide Web Conf., 1997.

[**Hap02**] Hapner M., Burridge R., Sharma R., Fialli J. e Stout K.: *Java Message Service*. Sun Microsystems Inc., Santa Clara, CA, 2002.

[**Hol95**] Holbrook H., Singhal S. e Cheriton D.: *Log-based receiver-reliable multicast for distributed interactive simulation*. In Proceedings of ACM SIGCOMM'95, 1995.

[**Hua01**] Huang Y. e Garcia-Molina H.: *Publish/Subscribe in a mobile environment*. In Proceedings of MobiDE, 27-34, 2001.

[**Lew99**] Lewis R.: *Advanced Messaging Applications with MSMQ and MQSeries*. QUE, 1999.

[**Lin96**] Lin J. e Paul S.: *A reliable multicast transport protocol*. In Proceedings of IEEE INFOCOM'96, IEEE Society Press, Los Alamitos, CA, 1414-1424, 1996.

[**Liu03**] Liu Y. e Plale B.: *Survey of Publish Subscribe Event Systems*. May, 1999.

[**Omg01**] Object Management Group: *CORBA Event Service Specification*. Needham, MA, 2001.

[**Ora02**] Oracle: *Oracle9i Application Developer's Guide-Advanced Queuing*. Redwood Shores, CA, 2002.

[**Pow96**] Powell D.: *Group communication*. Commun., ACM 39, 4 (Apr.), 50-97, 1996.

[**Rat01**] Ratnasamy S., Handley M., Karp R. e Shenker S.: *Application-level multicast using content-addressable networks*. In Proceedings of the Third International Workshop on Networked Group Communication, 2001.

[**Sch94**] Schilit B., Adams N. e Want R.: *Context-Aware Computing Applications*. 1st International Workshop on Mobile Computing Systems and Applications, Santa Cruz, 1994.

[**Seg00**] Segall B., Arnold D., Boot J., Henderson M. e Phelps T.: *Content-based routing with Elvin4*. In AUUG2K (Canberra, Australia), 2000.

[**Ses97**] Sessions R.: *COM e DCOM: Microsoft's Vision for Distributed Objects*. John Wiley & Sons, New York, NY, 1997.

[**Tib99**] TIBCO: *TIB/Rendezvous*. White paper, TIBCO, Palo Alto, CA, 1999.

[**Tri04**] Triantafillou P. e Economides A.: *Subscription summarization: a new paradigm for efficient publish/subscribe systems*. Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), 2004.

[**Zhu01**] Zhuang S., Zhao B., Joseph A., Katz R. e Kubiatowicz J.: *Bayeux: An architecture for scalable and fault-tollerant wide-area data dissemination*. In Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'01), 2001.