UNIVERSITÀ
DEGLI STUDI
DI PALERMO

Un sistema parallelo per la valutazione di modelli di gestione della reputazione

Tesi di Laurea Magistrale in Ingegneria Informatica

V. Agate

Relatore: Prof. Giuseppe Lo Re
Correlatore: Ing. A. De Paola

UNIVERSITÀ DEGLI STUDI DI PALERMO
SCUOLA POLITECNICA

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

UN SISTEMA PARALLELO PER LA VALUTAZIONE DI MODELLI DI GESTIONE DELLA REPUTAZIONE

Tesi di Laurea di
Dott. Vincenzo Agate

Relatore:
Ch.mo Prof. Giuseppe Lo Re

Controrelatore:
Ch.mo Prof. Salvatore Gaglio

Correlatore:
Ing. Alessandra De Paola

Sommario

Molte applicazioni distribuite prevedono l'interazione di agenti sconosciuti, al fine di scambiare risorse e servizi; in tale scenario i sistemi di gestione della reputazione (*Reputation Management Systems* - RMS) rappresentano uno strumento utile a garantire un adeguato livello di affidabilità degli utenti coinvolti. A seconda dello specifico dominio applicativo, gli RMS presentati in letteratura hanno caratteristiche estremamente eterogenee e non esiste un approccio unico per valutarne e confrontarne le prestazioni, sia in termini di accuratezza nella stima della reputazione che di resistenza agli attacchi. Questo lavoro di tesi affronta tale problema presentando un nuovo sistema parallelo per la valutazione delle performance e della robustezza di un RMS indipendentemente dal dominio applicativo, fin dalla fase di progettazione. Il simulatore è pensato per offrire funzionalità che consentono di specificare in maniera semplice il modello di gestione della reputazione da valutare e che supportano il progettista nella simulazione degli scenari di attacco alla sua sicurezza. I risultati ottenuti indicano l'adeguatezza del sistema di simulazione progettato per gli scopi appena descritti, e la possibilità di ridurre sensibilmente i tempi necessari alle valutazioni sperimentali.

Indice

Introduzione

Internet oggi gioca un ruolo preminente nelle interazioni sociali, nel commercio e nello scambio di informazioni e servizi. La sua rilevanza nella vita quotidiana si concretizza in modi sempre più evoluti ed inimmaginabili fino a pochi anni fa. Le applicazioni, tramite cui gli utenti utilizzano la rete, sono caratterizzate dalla presenza di agenti software che cooperano al fine di risolvere problemi condivisi, pur mantenendo la propria autonomia ed individualità; in letteratura ci si riferisce a tali fattispecie con l'appellativo di ambienti distribuiti [20]. Internet, il web e la posta elettronica ne sono solo alcuni esempi pratici. Un tale contesto è l'ambiente ideale per la proliferazione di agenti eterogenei per interessi, obiettivi prefissati e comportamenti. In molti sistemi distribuiti, la richiesta di risorse e servizi richiede l'interazione con agenti sconosciuti che possono manifestare comportamenti imprevedibili. È plausibile aspettarsi che molti agenti, spinti dall'obiettivo di massimizzare la propria utilità, agiscano a detrimento del resto della comunità, con l'effetto di un degrado delle performance dell'intero sistema. Una delle modalità più efficaci di affrontare il problema è quello di affidarsi ad un sistema di gestione della reputazione (*Reputation Management System* - RMS), per incoraggiare gli agenti ad agire onestamente e in maniera cooperativa. In un ambiente totalmente distribuito, in cui è assente una autorità centralizzata capace di coordinare tutte le interazioni tra gli agenti, ogni membro della comunità può contribuire nella stima della reputazione degli agenti così da premiare i più meritevoli.
I modelli di gestione della reputazione, allo stato dell'arte, mostrano rilevanti differenze dovute alla eterogeneità dello scenario applicativo considerato, quali possono essere la condivisione di file in applicazioni peer-to-peer [18], framework per l'e-Commerce [5] e per le architetture orientate ai servizi [16, 2, 3].

L'idea alla base del progetto sviluppato, nasce dalla necessità di valutare le prestazioni di un nuovo sistema di gestione della reputazione, prima dell'effettivo rilascio del software, e dunque

già in fase di progettazione.
La scelta più comune fatta dai ricercatori è quella di progettare simulatori ad-hoc; tuttavia questa scelta costringe i progettisti a spostare l'attenzione su aspetti implementativi di scarso rilievo. Alcune soluzioni sono state proposte in letteratura, ma nessuna di queste permette di raggiungere gli scopi desiderati. Il lavoro di tesi qui presentato ha come obiettivo la progettazione un nuovo simulatore parallelo il cui fine è quello di supportare i ricercatori nella valutazione delle performance di un generico sistema di gestione della reputazione.

Il simulatore si basa sulla modellazione di un RMS come un algoritmo distribuito, che attraverso il paradigma di comunicazione tra processi e l'interfaccia di comunicazione *Message Passing Interface* abilita la sua esecuzione parallela su architetture di calcolo collegate in rete, abilitando la possibilità di una simulazione su larga scala. L'insieme di funzionalità offerte supportano il progettista nella definizione di questo modello e consentono la realizzazione di numerosi scenari di attacco specificandone il numero di agenti coinvolti e il loro comportamento. Il progettista infine, attraverso le funzionalità di raccolta dei dati, potrà analizzare l'efficacia e la robustezza del sistema di reputazione al termine della simulazione, osservando l'evoluzione e la reazione degli agenti coinvolti.

Struttura della tesi

Nella prima parte di questo lavoro di tesi si introducono i principali concetti teorici che motivano l'esistenza dei sistemi di gestione della reputazione. Essi nascono dall'esigenza di riconoscere ed impedire comportamenti antisociali in sistemi distribuiti multi-agente. Si farà riferimento al concetto di agente, proprio dell'Intelligenza Artificiale. Agenti con scopi eterogenei, interagiscono adottando strategie contrastanti. In ambienti distribuiti privi di entità centralizzate in grado di controllare le interazioni tra gli agenti, è necessario individuare e inibire comportamenti «parassita».
I comportamenti fraudolenti ad opera di agenti il cui intento è quello di massimizzare la propria utilità, anche a discapito del resto della comunità, trovano piena giustificazione nella teoria dei giochi, una scienza matematica ideata allo scopo di descrivere il comportamento razionale umano, brevemente accennata nel suddetto capitolo.
In ambienti popolati da agenti eterogenei, la tendenza alla cooperazione emerge in quegli agenti accomunati da obiettivi e strategie. Affinché il processo di cooperazione si realizzi, è necessario

che le entità si ritengano mutualmente affidabili. Nascono i concetti di fiducia e di reputazione. Si presenta in termini generali la definizione di sistema di gestione della reputazione attribuibile agli agenti software, ricordando che esso acquisisce significato se inserito in un contesto specifico. Viene mostrata una classificazione, riportata in letteratura, sulla base della provenienza delle informazioni utilizzate nella definizione di reputazione, con una trattazione dei principali approcci alla loro costituzione.

Viene successivamente presentata un'analisi dei sistemi di gestione della reputazione proposti in letteratura, delle principali vulnerabilità a cui risultano suscettibili e dei simulatori realizzati per effettuarne la sperimentazione.
I sistemi di gestione della reputazione in genere adoperano un meccanismo per la valutazione della fiducia, che in base all'origine delle informazioni per il suo calcolo, assume forme diverse. Il caso di studio individuato è un RMS costituito dalle principali componenti delle diverse soluzioni presentate in letteratura. La trattazione dei modelli sopra specificati risulta utile per individuare il metodo per la definizione delle specifiche progettuali e per la definizione delle principali componenti di un sistema di gestione della reputazione.
In particolare, le componenti comuni ad un generico sistema di gestione della reputazione risultano essere: il meccanismo di valutazione della fiducia, il protocollo di diffusione delle opinioni, la tecnica utilizzata per la fusione delle informazioni e il sistema di incentivi che ha come obiettivo quello di spingere gli agenti ad agire in maniera cooperativa.
La modellazione del RMS impone l'introduzione di una notazione rigorosa al fine di permettere di verificarne la sua correttezza formale e di fornire gli strumenti necessari a valutarne il costo computazionale; gli algoritmi distribuiti offrono tale modello. Il modello a cui ci si riferisce è il modello di rete sincrona, rappresentabile come un grafo non orientato, in cui i processi, a seguito di alcune assunzioni riguardo la loro temporizzazione, ne identificano i nodi.
La realtà teorica sopra discussa, deve concretizzarsi con l'adozione degli opportuni strumenti per la realizzazione del *framework* di simulazione. Viene, pertanto, riportata un'anteprima degli strumenti e delle tecnologie usate nell'area della programmazione di applicazioni per sistemi a memoria distribuita. Ci si focalizza in particolare sul paradigma di comunicazione *Message Passing Interface* (MPI), divenuto nel corso degli anni uno stantard de facto tra le *Application Programming Interface* (API) per la programmazione su sistemi a memoria distribuita.
Viene presentato il modello architetturale del *framework* di simulazione costituito da due diversi livelli di astrazione: il livello più alto responsabile della modellazione del sistema di gestione

della reputazione, e il livello di astrazione più basso responsabile della gestione dei processi coinvolti nella simulazione. Il simulatore fornisce all'utente molte funzionalità, che vanno dalla definizione del modello di gestione della reputazione, alla definizione della topologia della rete simulata e degli scenari di attacco al RMS.
Infine, vengono descritti dettagliatamente i passi necessari alla configurazione degli scenari per la simulazione degli attacchi, soffermandosi in particolare sulla realizzazione degli attacchi di *whitewashing* e di *slandering*, due tra i più comuni studiati in letteratura. Si mostra in che modo i dati raccolti dal simulatore possano essere utilizzati per la valutazione delle performance del sistema e per la definizione di opportune modifiche progettuali al sistema di gestione della reputazione, affinché il RMS risultante sia adeguato allo scenario applicativo.

Capitolo 1

Il quadro teorico

Uno degli obiettivi dei sistemi di gestione della reputazione è quello di riconoscere ed impedire comportamenti antisociali in sistemi distribuiti multi-agente. In tale scenario, agenti eterogenei tentano di perseguire i loro obiettivi, spesso in contrasto. Essere in grado autonomamente di bloccare eventuali contatti con agenti «parassita», special modo in ambienti privi di entità centralizzate in grado di controllare le interazioni tra gli agenti, diventa di vitale importanza per limitare i comportamenti fraudolenti.
Le motivazioni che spingono gli agenti ad agire in maniera egoistica, per massimizzare la propria utilità, anche a discapito del resto della comunità, sono pienamente giustificate dalla teoria dei giochi, una disciplina che ha lo scopo di descrivere il comportamento razionale umano. Gli elementi teorici portanti di questa scienza sono stati trasferiti nel contesto dei sistemi intelligenti, costituiti da entità il cui comportamento è per certi versi assimilabile a quello umano.
In un tale scenario, ogni agente interagisce con gli altri, cercando di perseguire il proprio obiettivo, che tipicamente consiste nel massimizzare una funzione di utilità. Ovviamente gli obiettivi possono essere contrastanti, e molto spesso la scelta di un agente limita i benefici degli altri.
Si vuole qui descrivere uno scenario in cui la maggior parte degli agenti adotta un comportamento comune di collaborazione e condivisione, mentre una piccola percentuale vuole di fatto sfruttare tale situazione adottando comportamenti opportunistici o nocivi nei confronti degli altri.

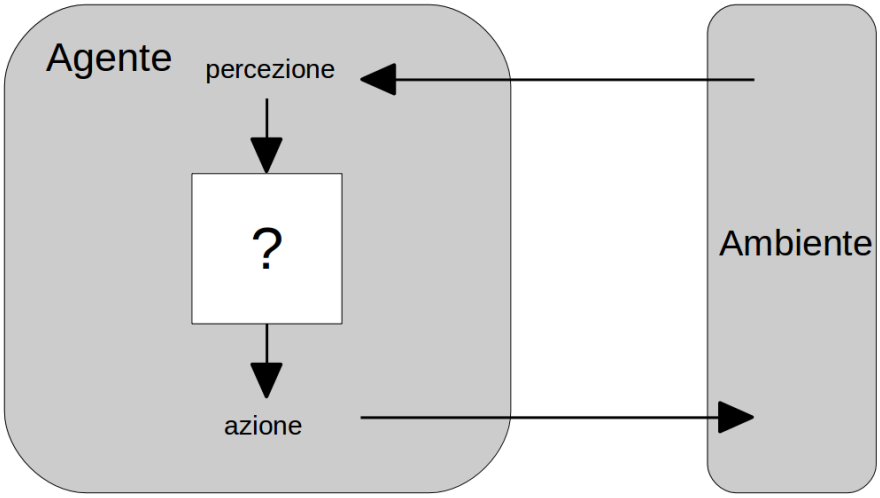

Figura 1.1: Flusso operativo agente. Immagine tratta da [19]

1.1 La teoria dei giochi

Un agente è una entità che esegue un compito operando autonomamente, percependo l'ambiente e perseguendo degli obiettivi. Secondo la teoria dei giochi, è da considerarsi razionale se agisce in modo da ottenere il miglior risultato o, in condizione di incertezza, il miglior risultato atteso. In genere, attraverso una fase di percezione, raccoglie degli input, che, a seguito di una elaborazione, restituisce all'ambiente. Un agente software, in particolare, può ricevere in input segnali, file o pacchetti di rete e può alterare l'ambiente restituendo segnali, file o pacchetti di rete [19]. Nella rappresentazione più classica il flusso operativo (figura 1.1) di un agente nel suo ambiente può essere riassunto dalle fasi di:

- percezione
- ragionamento
- azione.

La teoria dei giochi modella matematicamente sistemi composti da agenti, studiandone il comportamento razionale in condizione di interdipendenza strategica, ovvero quando la scelta dell'azione da prendere tiene conto delle scelte e delle reazioni degli altri agenti.
È uno dei pochi casi in cui una teoria matematica viene espressamente ideata per le scienze

sociali, con un rilevante impatto sugli ambiti più disparati come l'economia.
In questo tipo di rappresentazione, chiamata gioco, gli agenti coinvolti mirano a massimizzare una propria funzione di utilità. Il loro operato modifica l'ambiente e l'operato degli altri giocatori.
Se le decisioni di un giocatore si scontrano con le decisioni di altri si è in presenza di giochi non cooperativi. In tali scenari, un giocatore percepisce che adottare un comportamento non cooperativo può portarlo ad una situazione di maggiore guadagno.
Il modello sopra menzionato considera l'operato di un agente come totalmente razionale. In situazioni più vicine alla realtà , tale assunzione può a volte venir meno. La teoria dei giochi classica si occupa di situazioni che si concludono con una singola interazione tra giocatori perfettamente razionali i quali conoscono ogni dettaglio del gioco. Nella teoria dei giochi evolutiva, invece, si considera un gioco che si ripete continuamente nel tempo. Ognuno dei giocatori è programmato per usare una determinata strategia di gioco, ma il comportamento non è immutabile nel tempo; al contrario, esso e la sua distribuzione all'interno della popolazione sono modificati da un qualche processo evolutivo. Questo approccio permette ad un agente di affrontare situazioni in cui non ha una visione completa dell'ambiente che lo circonda, e i suoi avversari non prendono decisioni totalmente razionali. Nella teoria dei giochi evolutiva la migliore strategia non è tipicamente rappresentabile in un'espressione in forma chiusa, al contrario può essere utilizzato un meccanismo di apprendimento che possa garantire alla lunga una visione del mondo il quanto più possibile fedele. In ambienti popolati da agenti eterogenei, la tendenza alla cooperazione emerge in quegli agenti accomunati da obiettivi e strategie. Affinché il processo di cooperazione si realizzi è necessario che le entità si ritengano mutualmente affidabili. Nascono i concetti di fiducia e di reputazione.

1.2 Definizione di sistema di gestione della reputazione

I sistemi di reputazione rappresentano un ottimo strumento di classificazione senza il quale molte applicazioni distribuite non sarebbero riuscite ad affermarsi, dal momento che, in ambienti privi di informazioni certe, hanno reso possibile la valutazione dell'affidabilità di agenti sconosciuti. Questi schemi sono riconducibili ai meccanismi di relazioni umane e scambi intersociali.

Dunque, il concetto di reputazione, può essere attribuibile, non solo a persone fisiche, ma altresì a cose ed agenti software. È utile ricordare, che, come nelle interazioni sociali, il significato di reputazione è valido se inserito in un contesto di riferimento.
Le fonti dalle quali attingere le informazioni necessarie possono essere il frutto di una esperienza diretta, quindi già in possesso di chi effettua la valutazione, ma un contributo consistente alla determinazione della reputazione può provenire da esperienze esterne. A queste ultime è preferibile fare affidamento nel caso in cui siano assenti informazioni dirette. Alla luce di quanto detto, nella determinazione di un sistema di reputazione è necessario:

- definire gli obiettivi nel sistema;
- identificare le possibili fonti di opinioni;
- codificare le informazioni ottenute.

Nel caso in cui l'oggetto della valutazione sia una qualche caratteristica positiva dell'agente, per esempio il numero di transazioni eseguite o il numero di richieste soddisfatte, allora si definirà una reputazione positiva, al contrario, qualora si vogliano caratterizzare aspetti negativi, come il numero di richieste rigettate o il numero di transazioni non eseguite, si definirà una reputazione negativa.
Occorre infine fare una classificazione sulla reputazione in base alla provenienza delle informazioni che hanno portato alla sua costituzione. Una sua stima, determinata da interazioni derivanti da esperienze personali o esterne, ma in ogni caso dirette, danno vita alla così detta reputazione locale. Qualora le informazioni utilizzate per una sua stima, invece, siano il frutto di una conoscenza comune, si è in presenza di una reputazione globale. Sorge spontaneo chiedersi l'utilità dell'ultima classificazione, o cosa possa spingere un agente a farvi ricorso. Le ragioni di tale scelta vanno individuate nell'evidente costatazione che sia pur sempre preferibile avere una qualche forma di conoscenza, rispetto all'informazione nulla [4].

1.3 Sistemi di gestione della reputazione centralizzati e distribuiti

La collaborazione tra agenti assume significato qualora avvenga in situazioni di assoluta affidabilità e la reputazione rappresenta il suo unico strumento di misura.

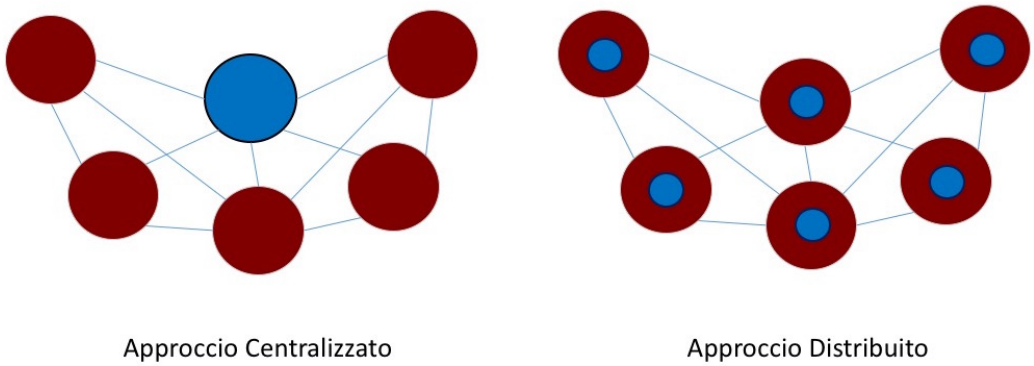

Figura 1.2: Confronto tra la struttura dell'approccio centralizzato e dell'approccio distribuito

Si pensi ad un ambiente distribuito rappresentato mediante un grafo in cui i nodi individuano gli agenti, e gli archi siano i canali di comunicazione attraverso i quali interagiscono.
È intuitivo pensare alla costituzione di un'Autorità Centralizzata (CA), figura 1.2, che si occupi della gestione dell'affidabilità degli agenti in gioco, che faccia in un certo senso da garante al di sopra di tutte le parti. Ma in uno scenario totalmente distribuito, in cui ogni agente in gioco è un nodo della rete, una tale scelta, risulta per certi versi impraticabile per via di differenti motivazioni. In primo luogo un'entità centralizzata rappresenterebbe sicuramente un elemento dell'architettura distribuita a cui cui tutti i nodi dovrebbero collegarsi, dovrebbe quindi offrire un servizio a qualsiasi nodo della comunità interessato. Dovrebbe inoltre avere caratteristiche, in termini di capacità computazionali e di meccanismi di difesa, superiori a qualsiasi altro nodo. La sua realizzazione, avrebbe un costo non indifferente, molto spesso insostenibile o non proponibile alla comunità. Inoltre, un qualsiasi tipo di disservizio della CA, non sarebbe tollerabile perché determinerebbe la totale assenza del servizio. Potrebbe essere più interessante realizzare la stessa attività, sostenendo un piccolo costo in termini computazionali per singolo nodo, in modo da eliminare il singolo punto di fallimento presente nell'approccio. L'onere computazionale, nell'approccio distribuito, sarebbe quindi a carico di tutti i nodi facenti parte della comunità, eliminando il collo di bottiglia dal sistema. Ogni nodo quindi dovrebbe collezionare informazioni sugli altri nodi con i quali è in contatto, stimando un valore di reputazione, e condividendo infine tale visione con la collettività.

Capitolo 2

Stato dell'arte

In questo capitolo viene proposta una panoramica sui sistemi di reputazione proposti in letteratura, sulle principali vulnerabilità a cui risultano suscettibili e sui simulatori proposti per la loro valutazione sperimentale.

2.1 Il sistema di gestione della reputazione

Diversi modelli di gestione della reputazione per sistemi distribuiti sono stati proposti in letteratura. Gli autori di [25] analizzano le componenti principali di un sistema di "*fiducia*", nel contesto dei Sistemi Multi Agente (MAS), identificando una fase di "*trust evaluation*", la quale valuta l'affidabilità degli agenti coinvolti nelle interazioni, e una fase di "*trust-aware decision making*" che usa valori di reputazione per selezionare gli agenti con i quali interagire. Gli autori classificano i metodi di valutazione della fiducia ("*trust*") in quattro classi:

1. Metodi basati sulle interazioni dirette passate;
2. Metodi basati sulle opinioni e testimonianze provenienti dagli altri agenti;
3. Metodi basati sulla conoscenza riguardo le relazioni sociali tra gli agenti;
4. Metodi basati su certificati forniti da autorità di terze parti.

Gli RMS per i sistemi distribuiti, dove una autorità centralizzata è mancante, appartengono alla seconda classe, secondo la quale ogni agente si basa su un protocollo distribuito per ottenere opinioni dagli altri agenti, unendo queste con le proprie passate esperienze in modo da ottenere la

reputazione di un dato agente. Questo schema è adottato da molti lavori presentati in letteratura. Lo studio di alcuni di questi può aiutare a capirne le principali funzionalità, le componenti fondamentali, e le caratteristiche comuni ad essi per lo sviluppo di un *framework* che sia in grado di modellarne uno generico.

2.2 EigenTrust

EigenTrust è un algoritmo di gestione della reputazione per le reti peer-to-peer (P2P) proposto in [13]. Nel corso degli anni è divenuto uno degli studi più citati in questo ambito. Di seguito se ne riassume il suo contenuto poiché, dal punto di vista logico, racchiude caratteristiche riscontrabili in molti sistemi esistenti.

2.2.1 Considerazioni progettuali

I suoi autori individuano almeno cinque problematiche da tenere in considerazione in qualsiasi sistema di reputazione:

1. *Self-policing*;
2. Anonimato;
3. Profitti ai nuovi arrivati;
4. *Overhead*;
5. Robustezza.

La prima delle problematiche vuole mettere in evidenza la necessità di escludere la presenza di un'autorità centralizzata per l'attività di controllo, ma di mantenere questo servizio a carico dei peer, al costo di un piccolo onere computazionale. La seconda prevede la garanzia dell'anonimato rispetto all'utente, ovvero l'associazione tra quest'ultimo e il valore di reputazione dovrebbe essere in qualsiasi modo scongiurato. Nei sistemi di reputazione proposti in letteratura, molto spesso, si tende ad assegnare automaticamente una etichetta positiva ai nodi appena collegati alla rete, favorendoli, di fatto incentivando comportamenti malevoli da parte di peer che vogliono ottenere lo stato di *newcomer*, e tutti i benefici derivanti. Un sistema di reputazione dovrebbe

comunque tenere in considerazione il carico in termini computazionali, di costo dell'infrastruttura, di messaggi scambiati. Infine dovrebbe garantire la resistenza alle collettività di nodi che tentino di sovvertire il sistema.

2.2.2 L'algoritmo

Sulla scia di altri lavori dell'epoca, il sistema di reputazione adottato si basa sull'utilizzo di una metrica data dalla differenza del numero di transazioni soddisfatte al netto di quelle non soddisfatte. Ogni peer *i* mantiene tale quantità, detta *Local Trust* per ognuno dei peer *j*.

$$s_{ij} = sat(i,j) - unsat(i,j)$$

La sfida, nel suddetto lavoro, è quella di trovare un modo per aggregare le informazioni per ottenere una visione quanto più omogenea possibile, generando il minimo *overhead* in termini computazionali e di comunicazione. Gli autori presentano un sistema che aggrega i valori di fiducia locali attraverso il concetto di "transitività" (transitive trust): un *peer* avrà un'alta opinione di quei peer che avranno fornito file autentici, di conseguenza riterrà più attendibili le informazioni, riguardo la reputazione, da essi provenienti. Il meccanismo utilizzato per mettere in atto il processo logico appena visto, si concretizza con l'autovettore principale sinistro della matrice dei valori di *Local Trust* normalizzati. Possiamo qui riassumere l'algoritmo nelle rimanenti fasi:

- Normalizzazione;
- Aggregazione.

Nella prima fase i valori s_{ij} subiscono una normalizzazione che li porta in un range compreso tra 0 e 1 attraverso la seguente equazione:

$$c_{ij} = \frac{max(s_{ij}, 0)}{\sum_j max(s_{ij}, 0)}.$$

Quest'ultima però presenta alcuni inconvenienti, come evidenziato dagli stessi autori. Il primo sta nell'incapacità di distinguere peer non collaborativi da peer appena collegati. Inoltre ad essi non è possibile attribuire un significato in senso assoluto, cioè è impossibile capire la bontà di un nodo da tali valori. Il risultato di questa fase è l'informazione pronta per il processo di diffusione nella rete.

Il processo di aggregazione, riportato nell'equazione in basso, permette la fusione delle informazioni attraverso una media pesata che utilizza come pesi i valori di reputazione del diffusore stesso:

$$t_{ij} = \sum_j s_{ij} s_{ij}.$$

Questi valori, fissato un agente j, al primo tentativo di stima, daranno informazioni solo sui vicini dei vicini. È importante notare che, se il processo di stima della reputazione viene eseguito per un numero di passi sufficientemente grande, è possibile ottenere una stima della reputazione per tutti i nodi della rete.

2.3 Gestione della reputazione nei sistemi distribuiti

Il caso di studio delle reti P2P non strutturate rappresenta un esempio lampante di ambiente distribuito afflitto da comportamenti antisociali. In [3] si propone un sistema di gestione della reputazione totalmente distribuito, nel quale gli agenti interagiscono autonomamente in modo da costruire un valore di reputazione su ogni partecipante. Lo scopo è quello di costruire una stima della reputazione che risulti più omogenea possibile tra gli agenti. I valori di reputazione stimati, sono usati in combinazione con un meccanismo di incentivi, il quale ha lo scopo di penalizzare gli agenti non cooperanti. La particolarità del sistema adottato sta nella capacità di adattare l'utilità percepita durante l'interazione tra gli agenti. Attraverso questa scelta, l'adozione di un comportamento cooperativo diviene una scelta vantaggiosa. Questo significa che gli agenti sono più motivati sia a condividere risorse, che ad ottemperare ai compiti relativi alla diffusione in rete delle richieste dei vicini. Più grande è il livello di cooperazione tra gli agenti, più grande è il valore di reputazione mantenuto dagli agenti che hanno cooperato con esso. Il meccanismo di incentivi privilegia gli agenti con alta reputazione, ai quali fornisce più risorse rispetto ai meno cooperativi. Gli agenti, considerati parassita (*Free Riders*), sono penalizzati vedendo rigettare le proprie richieste con una probabilità proporzionale alla loro cattiva reputazione. Nel meccanismo di gestione della reputazione in analisi, gli autori individuano due fasi: la prima per la costruzione di una stima locale e nella seconda dove le informazioni sulla reputazione sono inoltrate all'intera comunità. La prima fase sfrutta le informazioni collezionate durante le interazioni con gli agenti, integrando le informazioni passate con il contesto attuale. Durante la fase di diffusione, coppie di agenti mutualmente affidabili, scambiano i loro valori di reputazione, in tal modo ogni agente può integrare le proprie informazioni con quelle ricevute.

2.3.1 Stima locale della Reputazione

Questo sottosistema, come già anticipato, si occupa della stima del grado di cooperazione degli agenti con i quali ha interagito. Il suo ruolo è di cruciale importanza per il raggiungimento dell'equilibrio tra le risorse fornite da un agente e quelle da esso richieste. In particolare gli autori usano la differenza tra le risorse fornite rispetto a quelle ottenute, avvantaggiando in tal modo entrambi gli agenti, in situazioni di puro equilibrio, o il più generoso altrimenti. In maniera opposta, gli agenti meno generosi saranno penalizzati da tale meccanismo. Nel suddetto lavoro, viene mantenuto un doppio valore di reputazione:

- *Bad* reputation;
- *Good* reputation.

In letteratura molto spesso sono proposti sistemi di reputazione con un singolo valore. Gli autori di tale approccio ne impiegano due per superare l'incapacità di distinguere agenti appena subentrati nella rete rispetto ad agenti parassita conosciuti. Inoltre tali valori sono inseriti in un meccanismo di apprendimento per rinforzo. Nel sistema proposto oltre ai download completati con successo, un ulteriore servizio è tenuto in considerazione nel computo della reputazione, ovvero il servizio di inoltro delle richieste altrui, essenziale in tali applicazioni. Ogni agente i mantiene un profilo per ogni altro agente j che è stato precedentemente contattato. Ogni profilo è costituito dal numero di richieste (req_{ij}) che l'agente i ha soddisfatto per l'agente j e il numero di servizi ($serv_{ij}$) ottenuti da j. I valori precedenti sono stati utilizzati per calcolare una stima dei valori locali istantanei della *bad* ($\hat{gr}$) e *good* ($\hat{br}$) reputation.

$$\hat{gr}_{ij} = \begin{cases} \max(1 - \frac{req_{ij}}{serv_{ij}}, gr_{ij}) & \text{se } serv_{ij} \geq req_{ij}, \\ 0 & \text{altrimenti.} \end{cases}$$

$$\hat{br}_{ij} = \begin{cases} \max(1 - \frac{serv_{ij}}{req_{ij}}, br_{ij}) & \text{se } serv_{ij} < req_{ij}, \\ 0 & \text{altrimenti.} \end{cases}$$

Attraverso un approccio di apprendimento per rinforzo, la reputazione(*bad* o *good*) è frequentemente aggiornata grazie ai valori istantanei nel seguente modo:

$$r_{ij}(t) = \alpha * \hat{r}_{ij}(t) + (1 - \alpha) * r_{ij}(t-1).$$

2.3.2 Diffusione della reputazione

Il protocollo di diffusione della reputazione, è stato realizzato dagli autori, con lo scopo di trasmettere le informazioni riguardo il comportamento dei partecipanti, attraverso vicini mutualmente affidabili, così da ottenere una visione della rete quanto più uniforme possibile. Questo processo, in qualche modo, rispecchia le interazioni sociali. Mediando le informazioni provenienti da agenti che si ritengono affidabili, la probabilità di diffusione di false informazioni cala drasticamente. Di seguito è mostrata l'equazione utilizzata per tenere in considerazione le informazioni degli agenti vicini. Ogni valore ricevuto è comunque pesato dalla fiducia che si ha di quell'agente.

$$r_{ij}(t) = (1-\beta) * r_{ij}(t-1) + (\beta) * \frac{\sum_{k \in K} gr_{ik}(t-1) * r_{kj}(t-1)}{\sum_{k \in K} gr_{ik}(t-1)}$$

$$K = \{k : gr_{ik}(t-1) \geq \tau\}.$$

In questo caso K rappresenta l'insieme degli agenti che si ritengono affidabili, τ rappresenta invece una soglia che delimita il confine tra agenti affidabili e non.

2.4 Attacchi ai sistemi di gestione della reputazione

Sebbene i sistemi di reputazione permettano una più efficiente collaborazione tra nodi che perseguono uno scopo comune, permettendo di individuare nodi il cui comportamento devia dal tradizionale, è possibile che il suo funzionamento venga minato da particolari comportamenti dannosi. In questa sezione, viene effettuata una disamina degli attacchi contro i sistemi di reputazione. In particolare si farà riferimento alla tassonomia proposta in [12].

2.4.1 L'attaccante tipo

Le caratteristiche che contraddistinguono un attaccante possono essere davvero molte. La natura aperta dei sistemi di reputazione porta ad assumere, spesso, che l'attaccante sia interno. Si tratta quindi di un agente che ha legittimo accesso al sistema e che è in grado di partecipare al processo cooperativo secondo le regole prestabilite. Se al contrario, esso non è autorizzato all'accesso, si è in presenza di un attaccante esterno. L'attaccante può essere motivato da vari moventi: qualora esso voglia manipolare i valori di reputazione per aumentare il proprio tornaconto, questo sarà etichettato come nodo parassita (*Selfish*), se il suo scopo è invece quello di danneggiare i benefici portati alla comunità, esso sarà etichettato come nodo malevolo (*Malicious*).
Un attaccante può agire da solo, oppure in coalizione. Entrambi gli scenari sono plausibili, ma nei sistemi di reputazione il secondo risulta essere il più rilevante visto il suo carattere subdolo. È infatti molto difficile rilevare un'azione ben orchestrata, sia a causa della possibilità di nascondere il comportamento del singolo all'interno della regione malevola, sia a causa del peso che quest'ultima guadagna all'interno della rete.

2.4.2 Gli attacchi

La classificazione di seguito riportata fa riferimento allo scopo perseguito dai sistemi di reputazione. Essi vogliono assicurare che la metrica utilizzata rifletta, nel modo più accurato possibile, le azioni intraprese dai partecipanti al sistema. Questo obiettivo non può essere raggiunto se i partecipanti possono migliorare ingiustamente la loro reputazione, o degradare quella altrui. Vengono identificate le seguenti classi di attacchi:

- *Self-Promoting* - Gli attaccanti manipolano la propria reputazione aumentandola ingiustamente;
- *Whitewashing* - Gli attaccanti, dopo aver abusato delle risorse del sistema, mirano a migliorare artificiosamente la loro reputazione. Una volta ripristinata, possono continuare a compiere azioni dannose;
- *Slandering* - Gli attaccanti manipolano la reputazione di altri nodi vittima, realizzando, di fatto, un'azione diffamatoria;
- *Denial of service* - Il servizio di diffusione e di calcolo della reputazione viene ostacolato dagli attaccanti.

Self-promoting

In questo tipo di attacco, come abbiamo già avuto modo di vedere, i conduttori riescono ad aumentare la loro reputazione in modo falso. È realizzabile principalmente in quei sistemi che nel computo della reputazione, considerano solo feedback positivi. Può essere condotto da singole unità o da gruppi coordinati. Nella sua forma più semplice, si manifesta attraverso la diffusione di falsi feedback positivi riguardo se stesso.
La mancanza di un sistema di autenticazione e di integrità dei dati, risulta essere la causa principe della sua insorgenza. Tuttavia, anche nell'eventualità che un meccanismo di prevenzione fosse presente, l'attacco sarebbe comunque riproducibile qualora una singola entità potesse manifestarsi attraverso identità multiple (*Sybil Attack*). La volontà di eseguire tale classe di attacco può generare collusione tra i partecipanti.

Whitewashing

Un nodo parassita, può sfruttare la generosità degli altri agenti, fino all'istante in cui il suo valore di reputazione non va al di sotto di una certa soglia. Da quel momento l'intera collettività lo vedrà come un "*bad node*", per cui sarà predisposta ad escluderlo dalla rete. Ma se il sistema lo permette, esso può ripresentarsi sotto altre identità, in modo da godere nuovamente delle risorse offerte. Avrà di fatto ripulito la propria reputazione. Una forte agevolazione, alla proliferazione di questo attacco, può essere individuata nella eccessiva semplicità con cui è possibile ottenere

nuovi pseudonimi.
Questo tipo di attacco è particolarmente efficace se, nel computo della reputazione, si tengono in considerazione solo feedback negativi. Tuttavia, anche i sistemi che fanno utilizzo sia di feedback positivi che di feedback negativi, ne sono suscettibili, se tengono in considerazione solo la storia a lungo termine dell'operato di un nodo. Un attaccante potrebbe pian piano costruire una solida reputazione, per poi condurre attacchi dannosi per brevi periodi di tempo, in modo tale da non pregiudicarla.
La sua efficacia può essere notevolmente incrementata se congiunto ad altri attacchi. Se condotto unitamente al *Self-promoting*, potrebbe prolungare l'effetto negativo del secondo. Qualora fosse impiegato in combinazione con lo *Slandering*, potrebbe permettere l'occultamento dell'identità dell'attaccante. Per limitare gli effetti, un sistema di reputazione dovrebbe trattare diversamente i nuovi ingressi nella rete, rispetto ai partecipanti già presenti.

Slandering

Nei sistemi che non autenticano l'origine dei feedback, può verificarsi questa classe di attacchi. Una o più agenti malevoli, alterano volutamente valori di reputazione di altri agenti, diffondendoli sulla rete. Questo comportamento, nelle relazioni umane, è assimilabile alla diffamazione.
Se l'attacco è condotto da un singolo nodo, l'effetto è abbastanza contenuto, e potrebbe non pregiudicare nessuna entità coinvolta. Ma se la sua conduzione è frutto di un'azione congiunta, messa in atto da una collettività di nodi, il suo effetto potrebbe essere fatale al nodo vittima. Il livello di reputazione del nodo vittima non rispecchierebbe più la realtà, con la conseguente estromissione dal processo di collaborazione.
La vulnerabilità del sistema dipende fortemente dalla sensibilità della sua formulazione di reputazione ai feedback negativi. Maggiore è tale dipendenza, maggiore sarà il rischio di incorrere in tali attacchi. D'altro canto, se la sua sensibilità fosse bassa, i comportamenti diffamatori potrebbero essere consentiti per un tempo troppo lungo.
Per scongiurare tale fenomeno si potrebbe implementare un meccanismo di autenticazione, in modo da assicurare l'autenticità dei feedback, oppure pensare ad un sistema di reputazione che valuti solo le interazioni dirette tra i nodi.

Denial of Service

E' una classe di attacchi condotta da nodi malevoli non razionali. Il loro scopo è quello di sovvertire l'intero sistema di reputazione, impedendone il suo operato. Sono particolarmente suscettibili i sistemi di reputazione basati su un'autorità centralizzata.
La conduzione di tali attacchi è essenzialmente messa in atto sovraccaricando di richieste l'entità garante del servizio.

Le classi di attacco precedenti, possono incrementare i loro effetti dannosi se condotte in maniera ben organizzata. Non è necessario impiegare una strategia originale, né una ben definita. Gli attaccanti possono agire in gruppo, utilizzando strategie multiple, mutando comportamento nel corso del tempo. Se, i nodi collusi, diventano una quota significativa dell'intera comunità, allora l'intero sistema di reputazione può perdere la sua efficacia. Possono di fatto alterare la metrica di valutazione dell'intero sistema per accrescere i loro benefici.
Un attacco particolare, prende il nome di *Oscillation Attack*. I nodi collusi impersonano ruoli differenti in periodi alterni. Alcuni nodi potrebbero ad esempio mantenere un buon profilo e delegare i comportamenti dannosi ad altri, per poi scambiare i propri ruoli quando il sistema espone un principio di reazione. Questa strategia potrebbe passare del tutto inosservata, perché confusa con il comportamento ordinario.

2.5 Simulatori per la gestione della reputazione

Alcuni simulatori e *testbed* sono stati proposti per valutare le performance di un RMS, ma nessuno di essi permette simulazioni su larga scala del comportamento di un RMS sotto attacchi di sicurezza. Un testbed popolare di simulazione nel campo dei sistemi multi-agente è ART(*Agent Reputation and Trust*), proposto in [6]. ART permette di applicare diverse metriche di valutazione, e di definire competizioni nelle quali differenti strategie possono essere combinate e comparate in base al profitto ottenuto da ogni agente alla fine della simulazione. Tale caratteristica, utile in uno scenario MAS, è meno rilevante quando l'obiettivo è provare la capacità di un RMS di scoraggiare comportamenti malevoli nei sistemi distribuiti. TREET [14] permette di valutare RMS in uno scenario di "mercato" , misurando la resistenza del sistema con rispetto ad alcuni attacchi (*Reputation Lag Attack, Proliferation Attack e Value Imbalance*), ma non ne considera alcuni tipici come *whitewashing* e *self-promoting*. TREET supera molte

limitazioni tipiche di ART, permettendo agli agenti di entrare e uscire dinamicamente dalla simulazione, anche se la sequenza di tali eventi non può essere determinata dallo sperimentatore, ma è generata in maniera casuale.
Gli autori di [1] propongono un testbed generico per la valutazione di RMS. L'approccio proposto richiede che lo RMS sia modellato come una sequenza di trasformazioni di un grafo che rappresenti le transazioni e la fiducia tra gli agenti. Gli autori formulano alcune RMS esistenti secondo il loro nuovo modello e propongono le loro valutazioni sperimentali. Anche se questo simulatore permette di valutare la resistenza ad attacchi di auto-promozione e di *slandering*, non simula le interazioni degli agenti che sono cruciali per effettuare simulazioni su larga scala in cui gli agenti possono modificare il loro comportamento.

Capitolo 3

Il caso di studio

In questo capitolo verrà individuato un modello di sistema di gestione della reputazione. Questo ci permetterà di fare delle utili considerazioni per la costruzione del *framework* di valutazione di un generico modello di RMS.

Occorre individuare le componenti più comuni alla maggior parte degli RMS per gli ambienti distribuiti proposti in letteratura, alcuni dei quali mostrati nel precedente capitolo, in modo da progettarne uno generico. La Figura 3.1 riassume schematicamente le componenti essenziali:

- un meccanismo di valutazione della local trust (*local trust evaluation*), usato per stimare il comportamento degli agenti coinvolti nelle interazioni dirette;
- un protocollo di diffusione (*gossip protocol*), che propaghi le informazioni agli altri agenti della rete;
- un meccanismo di fusione delle informazioni (*information fusion*), per agglomerare le informazioni raccolte attraverso il protocollo di gossip unendole alla local trust, per ottenere i valori di reputazione;
- un meccanismo di incentivi (*incentive mechanism*) che sfrutti i valori di reputazione in modo da scoraggiare comportamenti antisociali.

In questo lavoro consideriamo, come caso di studio per provare l'efficacia del *framework* nella valutazione di un RMS agli attacchi alla sua sicurezza, un RMS che includa tutte queste componenti e ispirato a [13] e [3].

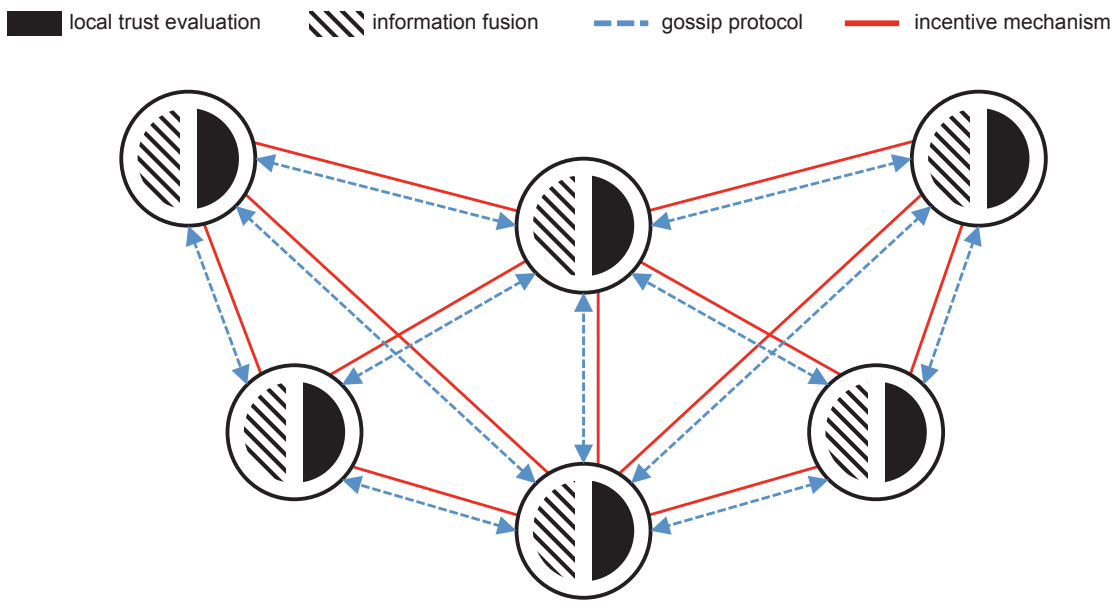

Figura 3.1: Struttura di un generico sistema di gestione della reputazione

3.1 Stima locale della fiducia

In questa sezione viene descritta la fase di stima locale della fiducia. L'obiettivo, per un nodo che si accinga ad espletare questa funzione, è quello di ottenere un giudizio sul grado di collaborazione dei nodi vicini, ovvero tutti quei nodi con i quali ha scambiato dei servizi. Genericamente, i servizi sui quali basare la valutazione, possono essere i più svariati, e cambiare in base al tipo di applicazione su cui si vuole realizzare il sistema di reputazione: per esempio in una rete *peer-to-peer* per la condivisione di file potrebbe essere il numero di file ottenuti, per un sistema di commercio elettronico il numero di transazioni correttamente espletate.
Il meccanismo di stima della fiducia locale qui scelto è una variante di quello proposto in EigenTrust [9], descritto nel capitolo precedente. Ogni agente i mantiene un profilo per ogni suo vicino j, memorizzando il numero di transazioni soddisfacenti, $sat(i,j)$, e insoddisfacenti, $unsat(i,j)$. La local trust s_{ij} è definita come la differenza tra questi valori, $s_{ij} = sat(i,j) - unsat(i,j)$.
Questi ultimi devono subire un processo di normalizzazione. Verrà, così, più comodo fonderle successivamente con le informazioni provenienti da altri nodi. A tali valori, contrariamente alla proposta originale, sarà possibile attribuire un significato in senso assoluto, eliminando alcuni

inconvenienti messi in evidenza dagli stessi autori. La normalizzazione qui proposta è ottenuta scalando s_{ij} in base al massimo valore osservato e tagliando fuori i valori negativi come segue:

$$c_{ij} = \frac{max(s_{ij}, 0)}{max_j(s_{ij})}.$$

Si ottengono pertanto valori scalari compresi tra 0 e 1, a cui è possibile attribuire il significato di reputazione r_{ij} nel caso in cui siano assenti informazioni provenienti da altri nodi. Tanto più il valore di *local trust* normalizzato è vicino allo 0, tanto minore sarà la fiducia che il nodo *i* riporrà in *j*, al contrario, tanto più è vicino al valore 1, tanto maggiore sarà il grado di fiducia riposto in *j*.

3.2 Protocollo di gossip

Il processo appena visto, seppure mantenga la propria utilità autonomamente, se unito ad un meccanismo di diffusione delle informazioni, consente di stimare più accuratamente la reputazione degli agenti coinvolti nel sistema.
Il protocollo di diffusione della reputazione, o protocollo di gossip, ha lo scopo di diffondere, tra vicini mutualmente affidabili, una visione il più omogenea possibile del resto della rete. Si vuole quindi scegliere un protocollo che ricalchi le normali interazioni umane. Quando due agenti si stimano a vicenda, possono scambiarsi informazioni circa l'affidabilità di altri nodi. Se uno possiede informazioni approssimative su alcuni vicini, può avvalersi delle informazioni di coloro di cui si fida per completarle, se ne possiede di false, può correggerle. È proprio questo il principio che si vuole sfruttare.
Al fine di non far perdere di genericità il RMS, si è scelto un protocollo semplice, per la diffusione delle informazioni:

- *ogni nodo* i*, diffonde a tutti i vicini i valori di reputazione calcolati per ogni vicino* j*, la cui valutazione sia frutto di esperienze dirette prima, e indirette dopo.*

Il protocollo di *gossip*, eseguito ad ogni step, permetterà ad ogni agente di avere il punto di vista dei suoi vicini riguardo la reputazione dei nodi di interesse. La scelta di un protocollo così semplice, non è in contrapposizione con il principio della mutua fiducia precedentemente enunciato: in questo caso infatti si è preferito delegare l'onere della valutazione delle informazioni affidabili al processo di "*information fusion*" che vedremo a breve, mantenendo la fase attuale

deliberatamente scarna da ogni ulteriore appesantimento, generando solo ed esclusivamente la pura trasmissione delle informazioni in possesso.

3.3 Fusione delle informazioni

Assumendo che un vicino si comporti in maniera corretta e razionale, riguardo la fornitura dei servizi, non è privo di significato pensare che esso mantenga lo stesso comportamento durante la fase di diffusione di informazioni riguardo l'affidabilità di altri vicini, anche perché un comportamento contrario andrebbe contro i propri interessi. Al contrario, è probabile, che un nodo con una cattiva reputazione, abbia interesse nel diffondere false informazioni per incrementare la propria reputazione relativa. Per tale motivo l'individuazione di informazioni affidabili deve avvenire sulla base della reputazione della fonte stessa.
Un nodo *i* si ritrova a questo punto a possedere informazioni provenienti dalle interazioni dirette con i nodi in questione, e informazioni provenienti da altri nodi. Diventa critica la scelta di un meccanismo che permetta la fusione di queste informazioni. Il meccanismo qui adottato si rifà al lavoro proposto in [3]. Ogni agente accoglie solo informazioni provenienti da agenti affidabili, ad esempio quelli la cui reputazione è sopra una data soglia τ, il cui valore va valutato sperimentalmente. Le informazioni agglomerate, sono pesate con la reputazione degli agenti dai quali esse provengono, e il risultante valore di reputazione r_{ij} è una combinazione lineare di questa media pesata e del valore normalizzato di fiducia locale, come specificato dalla seguente equazione:

$$r_{ij}(t) = (1-\beta) * c_{ij}(t) + \beta * \frac{\sum_{k \in K} r_{ik}(t-1) * r_{kj}(t-1)}{\sum_{k \in K} r_{ik}(t-1)},$$

dove beta è un coefficiente compreso tra [0,1] e K è l'insieme degli agenti affidabili:

$$K = \{k : r_{ik}(t-1) \geq \tau\}.$$

La adozione di una somma pesata tra la stima locale della fiducia e il valore medio delle informazioni riportate dagli altri è una soluzione comune in letteratura [23, 24].
Il processo di fusione delle informazioni può essere ripetuto ad intervalli temporali piccoli a piacere. Di fatto, il risultato è quello di aggiornare il valore di reputazione posseduto. Se la natura dell'applicazione è caratterizzata da una estrema dinamicità, diventa essenziale osservare

prontamente il reale grado di cooperazione dei nodi vicini, per cui è plausibile un aggiornamento più frequente del valore di reputazione.

3.4 Il meccanismo di incentivi

Il calcolo della reputazione di un nodo, rimane fine a se stesso, se non integrato con un meccanismo di incentivazione che garantisca che i nodi cooperativi sperimentino una qualità del servizio superiore ai nodi meno cooperativi. Il comportamento di un agente cooperativo, potrebbe essere premiato, garantendogli di ottenere i servizi di cui ha fatto richiesta, con una probabilità proporzionale alla sua reputazione, come descritto in [2]. Tale soluzione è uno degli approcci più popolari per implementare il sistema di incentivazione [25]. Un nodo che avrà un alto valore di reputazione vedrà soddisfatte le proprie richieste con alta probabilità; al contrario, un nodo con basso valore di reputazione, vedrà scartate le proprie richieste con alta probabilità.

Capitolo 4

Progettazione del Simulatore

4.1 Algoritmi Distribuiti - Elementi teorici

Gli algoritmi distribuiti sono algoritmi progettati per essere eseguiti su hardware composto da numerosi processori interconnessi. Parti di un algoritmo distribuito sono eseguiti contemporaneamente ed indipendentemente, ed ognuno ha accesso solo ad una quantità limitata di informazioni. Si presuppone che tali algoritmi lavorino correttamente anche negli scenari caratterizzati da velocità di elaborazione e di trasmissione delle informazioni eterogenee.
Essi giocano un ruolo fondamentale in una vastissima varietà di applicazioni, incluse le telecomunicazioni, i processi di distribuzioni delle informazioni, il calcolo scientifico e il controllo real time dei processi. E' importante che l'algoritmo sia efficiente e corretto formalmente. Tuttavia, poiché il loro utilizzo può assolvere compiti davvero complicati, la loro progettazione può diventare un processo estremamente difficoltoso. Per questo motivo è essenziale introdurre un modello e una notazione rigorosa che permetta di verificare la correttezza di un programma e di fornire gli strumenti necessari a valutarne il costo computazionale.

4.1.1 Il formalismo

In [17] vengono classificati tre modelli suddivisi sulla base del tipo di temporizzazione:

- Il modello sincrono;
- Il modello asincrono;

- Il modello parzialmente sincrono.

Il modello sincrono

È il più semplice da descrivere, sul quale ragionare e anche il più semplice da progettare. In questo modello i processi compiono passi in maniera simultanea, quindi la loro esecuzione avviene in round sincroni. Ovviamente è anche il modello che più si discosta dalla realtà dei sistemi distribuiti. Implementarlo nei sistemi distribuiti reali può risultare oneroso e inefficiente. Risulta comunque utile studiare la risoluzione di un problema in tale modello per capire come affrontarlo nei sistemi più realistici.

Il modello asincrono

I processi compiono i propri passi in ordine e velocità relative puramente casuali. Sebbene questo modello sia molto semplice da descrivere, è molto difficile da programmare per via dell'incertezza del verificarsi degli eventi. Il progettista è comunque svincolato dall'onere di dovere effettuare considerazioni legate alla temporizzazione. Gli algoritmi progettati risultano generali e portabili, spesso anche essi non corrispondono alle esigenze pratiche dei sistemi distribuiti reali.

Il modello parzialmente sincrono

È il modello più realistico e il più difficile per la progettazione degli algoritmi distribuiti. Similmente al primo modello, vengono fatte delle assunzioni sulla temporizzazione di eventi, ma l'esecuzione non avviene in passi bloccanti. Gli algoritmi prodotti secondo questo modello risultano molto efficienti, ma fragili se vengono meno le assunzioni fatte.

Modello di Rete Sincrona

Una rete sincrona è una collezione di processi comunicanti per mezzo di un sottosistema di comunicazione. In questo modello, il tipo di comunicazione più frequente è quella *point-to-point*, un tipo di comunicazione che si svolge tra due processi della rete per mezzo di alcune azioni di *send* e *receive*. Altri tipi di comunicazione possono essere utilizzate, come la *multicast* e la *broadcast*, che permettono rispettivamente di inviare messaggi ad un sotto insieme di processi e

all'intero insieme di processi che condividono il mezzo di propagazione del messaggio.
In questo modello, occasionalmente, si vogliono considerare sistemi in cui i processi possono iniziare la loro esecuzione in *round* differenti. Tale situazione può essere realizzata aumentando il grafo di rete attraverso uno speciale nodo, dal quale partono connessioni verso tutti gli altri nodi ordinari. Il suo ruolo è quello di inviare speciali messaggi di *wakeup* ai processi della rete che si trovano in uno stato di quiescenza, ovvero processi che non producono output sulla rete e che possono solo cambiare il loro stato al ricevimento di un messaggio di *wakeup*.
Al fine di definire formalmente una rete sincrona, possiamo immaginarla come un grafo orientato $G = (V, E)$. Utilizziamo la lettera n per denotare la cardinalità di $|V|$, ovvero il numero di nodi della rete. L'insieme E rappresenta l'insieme degli archi che connettono i nodi del grafo. Per ogni nodo i di G, utilizziamo $out\text{-}nbrs_i$ per indicare i vicini di i in G a cui arrivano i messaggi, e $in\text{-}nbrs_i$ per indicare i vicini di i in G da cui arrivano messaggi. Qualora volessimo considerare un grafo non orientato, ovvero un grafo in cui le connessioni sono bidirezionali, utilizziamo la notazione $nbrs_i$ per indicare i vicini di i in G. In questo caso i messaggi possono partire ma anche arrivare dai suddetti vicini.
Associamo un processo ad ognuno dei nodi del grafo G, e possiamo immaginare che essi possano comunicare sopra un canale associato ad ognuno degli archi monodirezionali del grafo. Infine chiamiamo M l'insieme di tutti i messaggi.

Il sistema *Send/Receive*

Il processo associato ad ogni nodo i è modellato come un'automa P_i. Solitamente riceve degli input e restituisce degli output. Gli input che riceve dagli altri nodi della rete sono della forma $receive(m)_{ji}$, dove j è il nodo mittente ed m è il messaggio. Gli output generati sono della forma $send(m)_{ij}$, dove j è un vicino destinatario del messaggio m. Questi sono gli elementi essenziali che caratterizzano un processo. Ovviamente, tutto il processo di elaborazione dei messaggi in ingresso per la produzione di messaggi di output rientra nella caratterizzazione del processo stesso.

4.1.2 L'algoritmo distribuito

Nel capitolo precedente è stato introdotto il sistema di reputazione scelto per questo lavoro di tesi. Abbiamo adesso tutti gli elementi teorici necessari per la sua definizione secondo il modello

degli algoritmi distribuiti.
Richiamando i concetti esposti precedentemente, per la realizzazione della simulazione, abbiamo bisogno di progettare un algoritmo unico, che venga eseguito da tutti i nodi della rete: supponendo che siano presenti le informazioni dirette relative alle interazioni con altri nodi, utili al calcolo della reputazione locale, possiamo rivolgere la nostra attenzione al meccanismo di comunicazione per la diffusione delle informazioni. Prima di vedere lo pseudo-codice dell'algoritmo è necessario fare alcune assunzioni riguardo la temporizzazione. La funzione *send* è intesa come primitiva di comunicazione non bloccante e asincrona. La prima caratteristica garantisce l'esecuzione delle istruzioni che seguono, dando la possibilità di effettuare ulteriori invii qualora siano necessari. Tale assunzione inoltre è utile a prevenire qualsiasi situazione di *deadlock* che si potrebbe venire a verificare altrimenti. La seconda caratteristica consente l'esecuzione della procedura a prescindere dallo stato del processo ricevente, infatti il processo ricevente potrebbe già essere in attesa attraverso la chiamata della procedura *receive* oppure non avere ancora raggiunto il suddetto istante per via di velocità di esecuzione relativa differente. La funzione *receive* è intesa come bloccante e asincrona. In questo caso è necessario arrestare il flusso di esecuzione fino a quando il dato in ingresso sia correttamente arrivato, affinché sia possibile utilizzarlo in maniera consistente. Sebbene l'utilizzo di routine bloccanti esponga il processo ad un maggiore rischio di *deadlock*, possiamo contare sul fatto che questo rischio sia scongiurato visto che il tipo di modello utilizzato è "*two-sided*", per ogni *send* effettuata da un processo mittente è presente una *receive* in attesa sul destinatario. La presenza di *matching* tra funzioni *send* e *receive* garantisce che l'attesa da parte delle primitive bloccanti sia comunque di durata limitata. Possiamo inoltre immaginare l'esecuzione continua dei vari passaggi dell'algoritmo di diffusione all'interno di un *round* che si perpetui per tutta la durata della vita del processo stesso. Riassumendo i passi salienti da effettuare all'interno di uno *step* sono:

- diffusione;
- raccolta;
- aggiornamento.

Lo pseudocodice mostra il flusso di esecuzione di ogni processo.
Nella prima parte dell'algoritmo vengono inizializzate tutte le variabili e i buffer necessari. In particolare m_{in} e m_{out} rappresentano il buffer di input ed il buffer di output, ovvero quelle locazioni di memoria dove il messaggio viene rispettivamente ricevuto nel primo caso, e pronto

Algorithm 1 Diffusione-Raccolta

```
1:  procedure PROCESSO
2:  inizialization :
3:      rep_vec ← alloc size of nbrs_i
4:      m_in ← alloc size of nbrs_i
5:      m_out ← alloc size of nbrs_i
6:      compute(rep_vec)
7:  loop:
8:      m_out ← rep_vec
9:      for j in nbrs_i do
10:         send(m_out)_ji.
11:     for j in nbrs_i do
12:         receive(m_in)_ji
13:     rep_vec ← update(m_in, rep_vec)
14:     goto loop.
15:     close;
```

per l'inoltro nel secondo. La variabile rep_{vec} rappresenta un vettore al cui interno troviamo i valori di reputazione dei nodi vicini. Durante la fase di inizializzazione tali valori sono calcolati attraverso la funzione $compute()$ che tiene in considerazione nel computo le esperienze derivanti dalle transazioni dirette con altri nodi. Tali valori, all'inizio di ogni round, vengono collocati nel buffer di output. A questo punto viene richiamata la routine di invio per ogni nodo appartenente ai vicini del nodo *i* preso in considerazione. Una volta conclusasi la fase di diffusione, si procede con la fase di raccolta delle informazioni provenienti dai processi vicini: per ognuno di essi infatti viene richiamata la routine *receive* che completerà il *matching* con la *send* del vicino. Al completamento della ricezione, il dato in ingresso viene utilizzato per l'aggiornamento dei valori di reputazione al fine di tenere in considerazione le informazioni provenienti dai vicini.

4.1.3 Analisi della complessità

Negli algoritmi distribuiti, solitamente, vengono tenute in considerazione diverse misure riguardo la loro complessità . Quella che qui verrà presa in considerazione è la *complessità di comunicazione*, misurata tipicamente in termini del numero totale di messaggi non nulli che sono inviati.

La *complessità di comunicazione* gioca un ruolo fondamentale, non solo perché un *overhead*

significativo potrebbe impedire il funzionamento dell'algoritmo stesso, ma soprattutto perché tipicamente i mezzi di comunicazione sono crocevia di una infinità di altre applicazioni distribuite che contemporaneamente immettono i loro messaggi. Garantire l'efficienza da questo punto di vista risulta quindi cruciale.
Prima di procedere alla valutazione della complessità di comunicazione, è bene fare alcune precisazioni: l'algoritmo in se non termina, ovvero la sua durata in termini di step è legata alla vita del processo stesso. Questo algoritmo è infatti solo una funzione ausiliaria rispetto ai compiti svolti dal processo, limitato solo alla diffusione e all'aggiornamento del valore di reputazione. Quindi calcolare il numero di messaggi totali non sarebbe ne prevedibile, nè significativo. Detto questo si possono tenere in considerazione il numero di messaggi inviati per ogni round. Le operazioni significative dell'algoritmo in questione sono le *send*. Il numero di tali operazioni k_i è uguale al numero di vicini posseduti dal nodo i. Volendo infine fare una stima globale, basterà tenere in considerazione il numero di nodi n presenti nella rete, ognuno dei quali produrrà k messaggi.

4.2 Message Passing Interface

Questa sezione presenta un'anteprima degli strumenti e delle tecnologie usate nell'area della programmazione di applicazioni per sistemi a memoria distribuita. Viene offerta una trattazione del paradigma di comunicazione *Message Passing Interface* (MPI) in riferimento a [21, 22], divenuto nel corso degli anni uno stantard de facto tra le *Application Programming Interface* (API) per la programmazione su sistemi a memoria distribuita. Si propone infine una trattazione più dettagliata delle primitive di comunicazione utilizzate per la realizzazione di questo lavoro.

4.2.1 Introduzione

Oggigiorno, la necessità di elaborare grandi quantitativi di dati, ha portato ad una costante richiesta di capacità computazionali sempre maggiori. Lo *High Performance Compuntig* (HPC) fornisce supporto per l'esecuzione efficiente di applicazioni avanzate. Esso rende possibili calcoli che precedentemente erano di difficile realizzazione. Le architetture moderne dei calcolatori spingono su un parallelismo a livello hardware, realizzando unità di esecuzione multiple, con il pipelining delle istruzioni e incrementando il numero di CPU. I calcolatori più potenti e veloci usano architetture a memoria distribuita e condivisa, dimostrando il trend, ormai affermato, dell'utilizzo di architetture ibride. Tra gli approcci a passaggio di messaggi più popolari vengono annoverati *Parallel Virtual Machine* (PVM) e MPI, il quale rappresenta il modello più utilizzato al momento. *Message Passing Interface* (MPI) è uno standard de facto per fornire ambienti HPC in cluster connessi attraverso interconnessioni ad alta velocità e Lan gigabit. Esso fornisce primitive per la comunicazione *point-to-point*, dando la possibilità ai processi di comunicare specificando l'identificativo del destinatario; offre alcune primitive per la comunicazione collettiva, semplificando la stesura di programmi; abilita la portabilità del codice, permettendo una semplice ricompilazione delle applicazioni realizzate; garantisce la possibilità di migliorare la scalabilità delle applicazioni, consentendo un'esecuzione ottimale all'aumentare della complessità dell'applicazione.
Supporta il modello di computazione parallela *Single Program Multiple Data* (SMPD). Durante il processo di calcolo, viene utilizzata la memoria locale.
Questo standard assicura la sua neutralità rispetto al tipo di architettura sottostante e l'indipendenza dal linguaggio di programmazione utilizzato. C e C++ sono le scelte più ampiamente adottate per implementare MPI, come in MPICH e LAM/MPI.

Più processi possono risiedere sulla stessa macchina fisica così come su un numero arbitrario di macchine. Per il trasferimento di dati tra processi, sono necessarie specifiche operazioni coordinate di invio e ricezione. Le modalità di comunicazione offerte da MPI sono *bloccanti*, *non-bloccanti*, *bufferizzate* e *sincrone*.

4.2.2 MPI

In principio l'elaborazione parallela era caratterizzata da dimostrazioni teoriche, sperimentazione e continue re-implementazioni per ogni nuovo calcolatore. Questa metodologia è sicuramente un ottimo modo per imparare a padroneggiare il calcolo parallelo, ma un pessimo modo per costruire l'infrastruttura necessaria a garantire stabilità e che possa diventare appetibile all'infuori del mondo accademico [21].
Divenne necessario un processo di standardizzazione che si realizzo a partire dal 1993. Un gruppo di rappresentanti delle industrie di computer, di enti governativi e accademici, unì le forze per lo sviluppo di una interfaccia standard per il modello di programmazione parallela "*message passing*". Questa organizzazione prese il nome di Message Passing Interface (MPI) Forum.
Un programma MPI è costituito da un insieme di processi in esecuzione, collegati logicamente da un canale attraverso il quale si realizza il passaggio di messaggi. Tali processi possono essere lo stesso programma (SMPD - *Single Program Multiple Data*) o programmi differenti (MPMD - *Multiple Program Multiple Data*).
Il modello architetturale della memoria MPI è un modello a memoria distribuita, in questo modo risulta impossibile ad un processo accedere a locazioni di memoria riservate ad altri processi, mentre, lo scambio di dati tra processi, avviene esclusivamente attraverso le *routines* MPI richiamate nei processi comunicanti. MPI definisce una libreria di *routines*, che implicitamente individua il modello di programmazione.
Le *routines* più importanti contenute nella libreria sono le cosiddette primitive *point-to-point*, che abilitano la comunicazione tra due processi specifici.
Lo standard MPI non è di per se un software, ma una specifica. Il suo contenuto è quindi una serie di regole che descrivono l'interfaccia di comunicazione, e non la sua implementazione.
Al fine di garantire maggiore efficienza nella sua realizzazione, non è stata fornita una implementazione, cosicché lo sviluppatore possa essere libero di realizzarne una propria, che si adatti alle sue esigenze. Per tale motivo non viene specificato alcun protocollo, né come i processi debbano essere creati o distrutti, in modo tale che ogni ambiente possa avere la sua implementazione.

Nella sua progettazione sono state tenute in considerazione alcune caratteristiche essenziali:

- portabilità ;
- efficienza;
- robustezza.

Una applicazione MPI dovrebbe richiedere solo la ricompilazione nel passaggio da una implementazione all'altra. In tal modo la progettazione di applicazione distribuita dovrebbe prescindere da qualsiasi sia la piattaforma su cui verrà realizzata. Dovrebbe inoltre supportare l'esecuzione su ambienti eterogenei. Una buona implementazione dovrebbe garantire il minimo spreco di risorse computazionali. Poiché il tipo di operazioni computazionalmente onerose sono le primitive di comunicazione, visto che utilizzano mezzi di propagazione come la rete e Internet, risulta cruciale realizzare applicazioni in cui il loro utilizzo sia ridotto all'essenziale. Dovrebbe fornire tutte le funzionalità utili nella pratica comune, e supporto per lo sviluppo di applicazioni e librerie parallele.

Diversi sono gli aspetti caratterizzanti un'implementazione di MPI che di fatto la rendono preferibile rispetto ad altre: una qualità desiderabile per lo sviluppatore di una applicazione è l'uniformità , ovvero la possibilità di potere eseguirla in qualsiasi ambiente con il minimo numero di modifiche al codice sorgente. Questa è una delle principali motivazioni che ha portato alla realizzazione di uno standard. Grazie ad esso, in teoria, uno sviluppatore può ricompilare il codice sorgente su qualsiasi ambiente che adotti un'implementazione di MPI.
Alcuni aspetti di MPI non sono individuati dalla specifica, la quale ne descrive solo il tipo di comportamento ma non l'implementazione: non si fa mai esplicito riferimento al quantitativo di memoria da riservare al *Buffering*; MPI non richiede che siano forniti servizi di input/output; non viene mai specificato come i processi debbano essere inizializzati e terminati.

Vale la pena valutare in che modo un'implementazione del protocollo a passaggio di messaggi si integri con l'ambiente in cui si inserisce; tale aspetto rappresenta uno dei più cruciali per la sua usabilità: il sistema operativo potrebbe individuare l'esecuzione di un'applicazione MPI come un insieme di processi indipendenti o come l'esecuzione di una singola entità distinta; l'implementazione potrebbe o meno fornire la flessibilità nel collegamento tra *stdin* e *stout*; la gestione dei processi potrebbe avvenire in modo più o meno robusto circa la loro terminazione, il rilevamento di processi orfani, in maniera del tutto automatica.

Infine è necessario tenere in considerazione un ulteriore criterio: tra una implementazione e l'altra non devono esistere significative differenze di performance a parità di hardware. Un sostanziale difetto di velocità potrebbe indicare scarsa compatibilità con il sistema operativo ospitante, o un'implementazione di scarsa qualità .

4.2.3 MPICH

Una delle implementazioni di maggiore successo, nonché scelta per lo sviluppo di tale tesi, è MPICH, implementazione portabile, multipiattaforma, liberamente disponibile e sviluppata dall'*Argonne National Laboratory* e dalla *Mississippi State University*. Essa stessa ha giocato un importantissimo ruolo nello sviluppo di MPI [11]. Ha dato vita a molte delle implementazioni commerciali di MPI. Tra queste si possono annoverare le implementazioni proposte dalla *Digital*, *Sun*, *HP*, *SGI/Cray*, *NEC* e *Fujitsu*. Alcune di queste sono ancora vicine ad MPICH, mentre altre, grazie ad un notevole sviluppo, hanno subito numerosi mutamenti.
La prima versione di MPICH venne scritta durante il processo di standardizzazione. L'esperienza dei suoi autori venne messa a disposizione per fornire importanti feedback. Il rilascio della prima versione avvenne contestualmente al rilascio della prima versione dello standard.
La sua portabilità è frutto della progettazione su due strati: la maggior parte del codice è "*device indipendent*" e implementato al di sopra di uno strato astratto chiamato *Abstract Device Interface* (ADI), il quale nasconde molti dei dettagli specifici, permettendo di fatto facilmente la sua portabilità in nuove architetture. Nelle versioni più recenti di ADI, vengono fornite nuove funzionalità , come l'esposizione di nuovi tipi primitivi (MPI *datatypes*), i quali permettono di astrarre totalmente i tipi di dato da utilizzare dallo specifico linguaggio di programmazione utilizzato o dalla architettura. Ogni implementazione di ADI prende il nome di "*device*". L'ultima versione rilasciata di tale implementazione, nonché la versione utilizzata per questo lavoro di tesi, è la *mpich-3.2*.

4.2.4 La Comunicazione Point-to-Point e i Messaggi

La comunicazione più elementare in MPI è la comunicazione "*point-to-point*", la quale abilità lo scambio di messaggi tra due processi, di cui uno è il mittente e l'altro il destinatario. In genere, il processo sorgente ed il processo destinazione, operano asincronamente. Ad esempio l'invio e la ricezione di un messaggio possono avvenire in instanti temporali differenti. Il pro-

cesso sorgente potrebbe completare l'operazione di invio prima che il destinatario richiami la routine per la ricezione, oppure il processo destinazione potrebbe iniziare la ricezione di un messaggio prima che questo sia stato inviato dal sorgente. Poiché il processo di invio e ricezione non sono solitamente sincroni, i processi potrebbero avere uno o più messaggi in attesa di ricevimento e prendono il nome di messaggi pendenti. In MPI la comunicazione è di tipo "*two-sided*", il che significa che sui processi coinvolti devono essere esplicitamente richiamate le routine di *send* e *receive*, in assenza delle quali, il messaggio, non viene trasmesso. Un messaggio consiste di due parti:

- l'intestazione;
- il corpo;

L'intestazione è del tutto simile alla busta che avvolge una lettera fisica, ed in esso sono indicati i seguenti elementi:

1. *source* - indica l'identificativo del processo mittente;
2. *destination* - indica l'identificativo del processo destinatario;
3. *communicator* - specifica il gruppo di processi alla quale sorgente e destinazione appartengono;
4. *tag* - indica la classe a cui appartiene il messaggio.

Il corpo del messaggio contiene le informazioni da trasmettere ed è caratterizzato dai seguenti elementi:

1. *Buffer* - individua l'indirizzo della locazione di memoria dove il dato uscente può essere trovato o il dato entrante deve essere salvato;
2. *Datatype* - individua il tipo di dato da inviare;
3. *Count* - individua il quantitativo di dati da inviare.

4.2.5 La comunicazione Bloccante e Non-bloccante

Una operazione di invio o di ricezione potrebbe essere di tipo bloccante o non bloccante. Una send o receive bloccante interrompe il flusso di esecuzione del programma, fino all'avvenuto completamento dell'operazione in corso. Con una send di tipo bloccante, ad esempio, la variabile che contiene il dato in uscita, può essere sovrascritta in maniera sicura, perchè si ha la certezza che il dato si già arrivato a destinazione o sia stato copiato in un buffer interno e inviato successivamente in background.
Una send o receive non bloccante restituisce immediatamente il controllo del flusso di esecuzione al programma principale, in tal modo il processore può continuare ad effettuare altre operazioni. In un secondo momento è possibile controllare l'avvenuta esecuzione dell'operazione. In questo caso, non si è certi del momento in cui l'operazione si sia conclusa, quindi sovrascrivere una variabile contenente il dato da inviare, potrebbe corrompere l'operazione stessa.

4.2.6 La Comunicazione Colletiva

MPI offre la possibilità , non solo di realizzare comunicazioni di tipo *point-to-point*, ma di richiamare routine che realizzano comunicazioni di tipo collettivo. Solitamente sono più utili nel caso in cui sia necessario lo scambio di messaggi all'interno di un gruppo di processi, riducendo la possibilità che si verifichino errori sostituendo più chiamate a primitive *point-to-point*, e aumentando la leggibilità del programma. Alcuni esempi sono l'operazione di *broadcast*, di *gather* e *scatter*, e di *riduzione*.

Broadcast

Questa primitiva di comunicazione permette l'invio di una copia del messaggio a tutti i processi facenti parte dello stesso gruppo.

Gather e Scatter

L'operazione di scatter permette la distribuzione di parti di dati, che inizialmente risiedono sullo stesso processo, a processi differenti. I dati, dopo avere subito l'elaborazione necessaria, vengono ricollezionati dal processo principale attraverso l'operazione inversa, gather.

Riduzione

E' una operazione che permette la concentrazione di dati provenienti da più processi, in un unico processo.

4.2.7 Le Primitive Utilizzate

Adesso verrà fornita una disamina dettagliata delle principali primitive di comunicazione utilizzate all'interno di un programma MPI.

MPI_Send

Primitiva di comunicazione point-to-point di tipo bloccante. Alla chiamata di questa routine, il contenuto del messaggio viene copiato in un buffer interno in attesa che possa essere recapitato al destinatario. Al completamento di tale operazione il flusso di esecuzione torna al programma chiamante.

MPI_Recv

Primitiva di comunicazione point-to-point di tipo bloccante. Alla chiamata di questa routine, il processo di esecuzione del programma chiamante viene fermato, in attesa dell'avvenuto completamento dell'operazione di ricezione.

MPI_Isend

Primitiva di comunicazione point-to-point non-bloccante. La chiamata di questa routine permette l'invio di dati ad un processo destinazione. La sua esecuzione non blocca il flusso di esecuzione del resto delle istruzioni del programma chiamante. E' utile quando devono essere effettuate più invii simultanei. Il contenuto della variabile da cui attinge il buffer non deve essere modificato. E' comunque possibile verificare il completamento dell'operazione attraverso la MPI_test.

MPI_Irecv

Primitiva di comunicazione *point-to-point* non-bloccante. L'invocazione di tale routine permette la ricezione di messaggi provenienti da un processo mittente. Anche in questo caso il flusso di esecuzione del programma chiamante non subisce interruzioni. Qualora fosse necessario utilizzare il dato contenuto nel messaggio, è necessario essere certi che questo sia stato ricevuto correttamente e che tale processo ci sia concluso. Per fare ciò è richiesto l'utilizzo delle routine per la verifica del completamento.

MPI_Wait

Routine di tipo bloccante posta in essere per la verifica del completamento di una *send* o *receive* di tipo non bloccante. Il suo utilizzo permette di arrestare il flusso di esecuzione del programma chiamante fintantoché la primitiva oggetto del controllo non sia terminata correttamente.

MPI_Test

Routine di tipo non bloccante per la verifica del completamento di una *send* o *receive* di tipo non bloccante. Il suo utilizzo permette di rilevare il completamento di un'operazione, cedendo immediatamente il controllo dell'esecuzione al programma chiamante, ritornando un valore vero o falso.

MPI_Probe

Primitiva non bloccante che permette la verifica della presenza di messaggi in entrata senza riceverli effettivamente. Sulla base delle informazioni da essa restituita, come il mittente, si può scegliere di scartare il messaggio in ingresso o riceverlo in un secondo momento.

MPI_Barrier

Può verificarsi la necessità che qualche processo non debba procedere oltre con la propria esecuzione fino a quando altri processi non abbiano compiuto le correnti istruzioni. La MPI_Barrier è una routine che abilita la sincronizzazione tra tutti i processi all'interno di un gruppo, bloccando l'esecuzione delle istruzioni fino al raggiungimento di tale istruzione da parte

di tutti i processi. Se da un lato è così realizzabile la sincronizzazione, dall'altro è bene tenere in considerazione che il suo utilizzo limiterà i benefici guadagnati in termini di velocità , dovuti al parallelismo. Infatti il tempo di esecuzione dell'intero programma sarà dettato dal processo più lento.

4.3 Progettazione del Simulatore

Lo studio di differenti sistemi di gestione della reputazione, ha messo in evidenza la totale assenza di uno strumento che garantisca la semplice e veloce realizzazione di una loro simulazione, per testarne le funzionalità , l'efficacia, e la resistenza agli attacchi più comuni. Si è deciso pertanto di colmare tale mancanza attraverso la realizzazione di un *framework* che sfrutti gli algoritmi distribuiti, fruibile a qualsiasi progettista che intenda realizzarne un proprio RMS, che si adatti all'esigenze dell'applicazione sulla quale costruirlo.
In questa sezione si mostrerà il processo relativo alla progettazione di tale *framework*, richiamando gli elementi di base presentati nelle sezioni precedenti, per la sua realizzazione.

4.3.1 Progettazione del simulatore parallelo

Valutare un sistema di gestione della reputazione è una sfida piuttosto complessa.
Uno studio teorico è senza dubbio propedeutico alla sua realizzazione, ma non sempre è possibile verificarne l'efficacia se non provandolo sperimentalmente. In tal senso gli sforzi di questo lavoro si sono concentrati sulla realizzazione di un ambiente di sviluppo.
L'obiettivo che si vuole raggiungere è quello di realizzare un *framework* che possa ospitare un generico RMS. Sfruttando le caratteristiche comuni alla maggioranza degli RMS, come la loro natura distribuita e il modello di comunicazione, si è pensato di realizzare un'infrastruttura solida sulla quale successivamente definire il proprio RMS e testarlo.
Prima di presentare il simulatore, risulta conveniente definire alcuni dei requisiti che esso debba soddisfare. L'ambiente che si vuole proporre deve essere in grado di simulare una grande varietà di scenari, come la rappresentazione di una topologia di rete generica e grande a piacere, o la conduzione di attacchi, diversi per tipologia e per scopo, su questa. Si evince quindi la necessità di costruire un sistema che abbia grandi possibilità in termini di scalabilità , è infatti desiderabile avere la possibilità di condurre esperimenti su larga scala, riponendo il minimo sforzo a carico dello sviluppatore. L'ideale sarebbe costruire un unico programma, che rappresenti il generico funzionamento di un nodo di rete e tutte le sue deviazioni, che possa essere istanziato un numero di volte tale da raggiungere il numero necessario per l'ottenimento di test attendibili. Pensare ad un unico calcolatore che possa gestire un cospicuo numero di nodi non è plausibile. L'*overhead* di comunicazione sarebbe tutto a carico del sistema operativo ospitante, e i requisiti di memoria

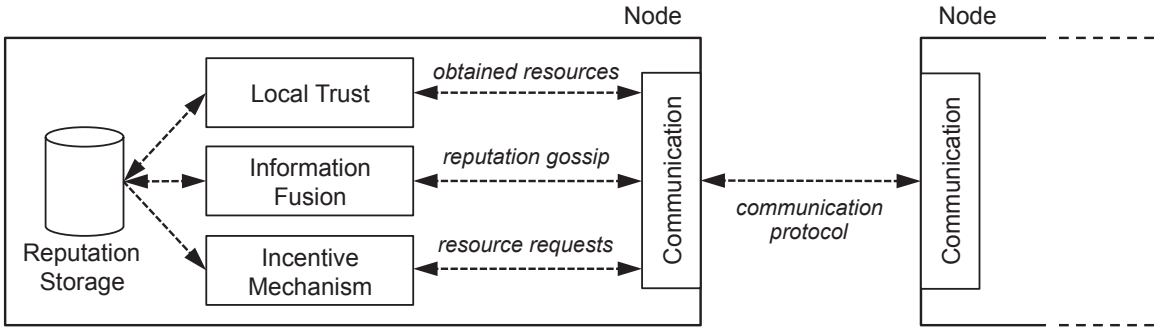

Figura 4.1: Modello di agente all'interno del *framework* di simulazione.

diventerebbero presto insostenibili per qualsiasi macchina commerciale. Tutti i requisiti appena menzionati sembrano indicare la necessità di realizzare un'applicazione per sistemi a memoria distribuita e il paradigma di comunicazione a *passaggio di messaggi* è ritenuta la scelta più opportuna.
Come visto nella sezione precedente, l'elemento principale in una tale applicazione è il processo, che può comunicare con altri processi attraverso una serie di primitive di comunicazione messe a disposizione dall'interfaccia MPI (figura 4.1). Il processo quindi incarna l'entità di rete generica, impersonandone tutti i comportamenti, compresi la soddisfazione e la richiesta di dati e servizi, e la realizzazione dei *task* relativi al sistema di gestione della reputazione. L'idea è quindi quella di associare ad un processo un agente del RMS, e grazie ad MPI assegnarlo ad un generico processore remoto che ne possa garantire la sua esecuzione parallelamente a quella dei suoi simili. Si realizza cosi la distribuzione dei processi nella rete, in tutte le macchine disponibili.

Il *framework* che si vuole proporre, lo si può idealmente pensare organizzato su due strati logici(figura 4.2):

- *reputation layer*;
- *simulation layer*;

Questa suddivisione, sebbene praticamente non sia tangibile, aiuta a comprenderne le funzionalità, permettendo di discernere funzioni di basso livello da quelle di alto livello. Un'architettura simile è stata adottata in [15], in cui si descrive la progettazione di un simulatore per le *Wireless Sensor Networks* in scenari distribuiti. Il *reputation layer* rappresenta il livello di astrazione più alto ed è costituito da un insieme di nodi interconnessi secondo una topologia scelta dal

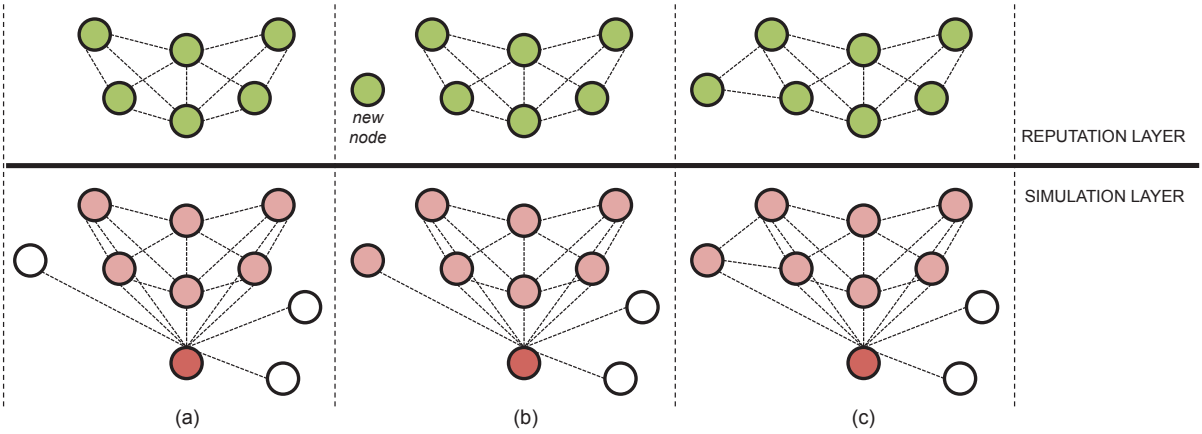

Figura 4.2: Suddivisione logica framework

progettista. Di fatto è la parte più interessante ai fini della simulazione poiché in essa viene mostrata la rete di agenti coinvolti nel sistema di reputazione. È bene sottolineare che a questo livello di astrazione, l'applicazione risulta:

- totalmente distribuita - ogni nodo infatti esegue la stessa sequenza di step contemporaneamente, su entità che possono risiedere in macchine fisiche disposte in qualsiasi luogo, sfruttando la rete per la comunicazione;
- non centralizzata - ogni agente mantiene le stesse caratteristiche degli altri, mantenendo la rete priva di qualsiasi figura centralizzata, che possa essere individuata come collo di bottiglia.

Vale la pena, inoltre, sottolineare la possibilità di fornire all'utente una serie di funzionalità per la configurazione che permettono di personalizzare la simulazione, come la possibilità di inserire nuovi nodi, permetterne la loro duplicazione, e caratterizzare scenari di attacco.
Allo strato più basso, invece, ogni agente è in realtà mappato su un processo differente, il quale grazie ad MPI, può essere inizializzato ed eseguito su una macchina reale disposta nella rete, e comunicante con gli altri attraverso la suddetta interfaccia. Qui, un processo particolare, chiamato *controller*, è responsabile della creazione di nuovi nodi del livello più alto, alterando il comportamento dei processi esistenti e già in esecuzione. Per essere più specifici, quando la simulazione è lanciata, un numero prestabilito di processi viene inizializzato e di conseguenza eseguito. Inizialmente tutti i processi avranno eseguito le stesse istruzioni, mentre successivamente, sulla base del proprio identificativo o *rank* in MPI, acquisiranno caratteristiche specifiche

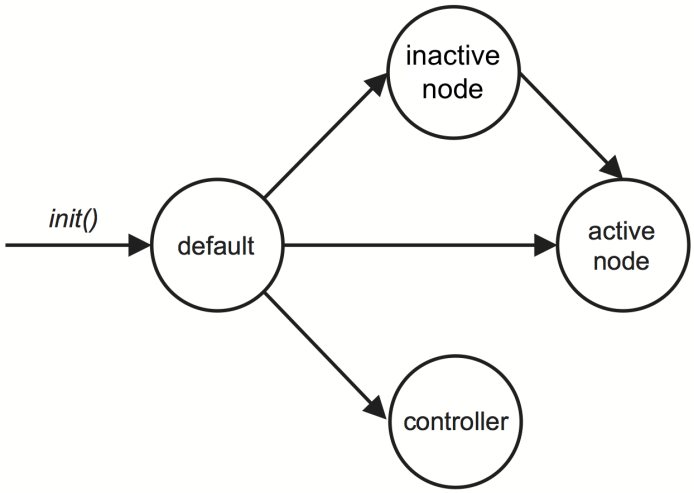

Figura 4.3: Generazione dei processi, derivanti dallo stesso programma

e ruoli differenti. compreso il processo *controller*. Il numero di processi è specificato dal progettista a seguito di una valutazione di massima sulla topologia di rete che intende simulare. La possibilità di una esecuzione parallela dipende ovviamente dalla disponibilità di processori liberi.

L'insieme dei processi in esecuzione, come è possibile vedere in figura 4.3, è dunque composto dal *controller*, dai processi *active*, i quali saranno responsabili della simulazione dei nodi di rete del livello più alto, e infine dai processi *inactive*, ovvero processi che sostanzialmente non eseguono istruzioni significative se non quelle che permettono ad essi di rimanere in ascolto nell'evenienza dell'arrivo di un eventuale messaggio. Ad onere del *controller* è infatti delegato il compito di variare la natura dei processi inattivi, attraverso l'inoltro di un messaggio, che di fatto, li rende attivi.
Nella figura 4.2 si può osservare in che modo i due strati subiscono l'inserimento di un nuovo nodo: al livello più basso, il processo *controller*, rappresentato in rosso, invia un messaggio, attraverso le primitive di comunicazione messe a disposizione dall'interfaccia MPI, ad uno dei processi inattivi raffigurati in bianco; quest'ultimo, rimane perennemente in ascolto, fino alla ricezione del messaggio che lo rende attivo (processi raffigurati in rosa); è di particolare interesse notare che al livello di astrazione più alto, la rete è totalmente ignara del coinvolgimento della figura centralizzata rappresentata dal *controller*, al contrario l'inserimento avviene in maniera del tutto trasparente, coinvolgendo solo i nodi vicini del nuovo arrivato.

4.3.2 Caratterizzazione della simulazione

Il primo passo nella caratterizzazione della simulazione è determinato dalla scelta del numero di nodi che si vogliono simulare e dalle loro interconnessioni. Viene quindi offerta la possibilità di inserire una topologia di rete generica attraverso l'inserimento di in file di configurazione, un file in cui vi è una rappresentazione di un grafo non orientato sotto forma di matrice di adiacenze. Ogni nodo inserito nella topologia, deve essere accompagnato da una etichetta, la quale indica il grado di cooperazione dello stesso definendone di fatto il comportamento. Tale etichetta è un valore numerico compreso tra 0 e 1: un nodo caratterizzato da un valore prossimo allo 0 soddisferà un basso quantitativo di richieste pervenute dai nodi vicini, al contrario un valore prossimo a 1, contraddistinguerà quei nodi con un alto grado di collaborazione. Entrambe le caratteristiche permettono da un lato la realizzazione di simulazioni generiche indipendentemente dal fatto che si voglia testare lo RMS su piccola o larga scala, dall'altro la creazione di scenari particolari per la conduzione di attacchi al sistema che vedremo a breve.
Nella valutazione locale della fiducia, come si è potuto vedere nel capitolo 3, sono tenute in considerazione il quantitativo di richieste e servizi soddisfatti dai nodi vicini. Il *framework* di simulazione permette di collezionare tali valori consentendone di variarne la granularità . Si potrebbe essere interessati a tenere in considerazione tutti i valori fin dall'inizio della simulazione, oppure al valore medio all'interno di una finestra temporale scorrevole. L'impostazione di una finestra di grandi dimensioni permette di tenere in considerazione la storia passata del comportamento dei nodi, sebbene diminuisca la capacità di rilevare variazioni istantanee. Al contrario una finestra di piccole dimensioni è più sensibile alle variazioni istantanee, ma tiene poco in considerazione la storia passata. Attraverso il simulatore è possibile condurre delle sperimentazioni al fine di individuare la dimensione ottimale della finestra, al variare dello scenario applicativo.
Il protocollo di gossip consente di definire il meccanismo di diffusione delle informazioni. Nel processo di caratterizzazione di un RMS risulta quindi utile potere personalizzare tale meccanismo, lasciando l'onere della realizzazione della comunicazione al simulatore. Lo sviluppatore quindi può scegliere il tipo di informazione da diffondere, sia essa la valutazione locale della fiducia o la reputazione globale, o scegliere a chi diffonderla. I destinatari potrebbero infatti essere tutti i nodi vicini del nodo mittente, ovvero tutti quei nodi con il quale ha interagito, o

tutti i nodi della rete, realizzando in tal caso una comunicazione collettiva.
Le informazioni trasmesse vengono raccolte dai nodi destinati. I dati grezzi vengono elaborati dall'algoritmo di fusione delle informazioni, il quale può essere definito dallo sviluppatore dello RMS. Si ritiene utile infatti dare la possibilità di personalizzarlo, affinché alle informazioni raccolte sia dato un peso proporzionale alla metrica che lo sviluppatore ritiene più opportuna. Le informazioni ad esempio potrebbero essere fuse attraverso una semplice media pesata al fine di ottenere reputazione globale, o essere inseriti opportunamente in un meccanismo di apprendimento.
Al *designer* infine è data la possibilità di specificare il meccanismo ad incentivi che ritiene più opportuno per il suo scenario applicativo. Il meccanismo di default prevede il rilascio di risorse e il soddisfacimento di richieste con una probabilità proporzionale alla reputazione globale del nodo destinatario e al grado di collaborazione del nodo mittente.

4.3.3 Conduzione degli attacchi

Uno degli aspetti più interessanti del *framework* di simulazione proposto è senza dubbio la possibilità verificare la resistenza di un RMS ad alcuni degli attacchi più comuni e di constatarne gli effetti sulla collettività . Come ampiamente discusso nel capito 2, i sistemi di gestione della reputazione, sono soggetti ad alcuni attacchi condotti da nodi maliziosi, motivati dall'intenzione di percepire un'utilità maggiore dal sistema o dal danneggiare insiemi o singoli elementi della comunità .
Il progettista può simulare uno scenario di attacco specificandone l'istante di inizio, la lista dei nodi coinvolti e il tipo di attacco. Il *framework* fornisce il supporto per due classi di attacco: lo *slandering* e il *whitewashing*. Il primo prevede la diffusione di false informazioni di reputazione con l'obiettivo di diminuire l'utilità percepita da uno o più nodi vittima. Per la simulazione di un attacco di tipo *slandering* (figura 4.4) è necessario selezionare uno o più nodi vittima V, e un insieme di nodi maliziosi M. Il simulatore provvederà a collezionare l'utilità percepita dai nodi coinvolti e il valore di reputazione degli stessi. Sarà quindi possibile effettuare opportune considerazioni sull'efficacia dello RMS sulla base dei dati ottenuti. La realizzazione di questo attacco avviene al livello di reputazione, tenendo in considerazione la suddivisione logica fatta precedentemente. In questo caso i nodi appartenenti all'insieme M, nel consueto processo di diffusione delle informazioni, alterano volutamente il valore di reputazione del nodo vittima inserendoli nel meccanismo di invio e ricezione delle informazioni. In particolare il contenuto del

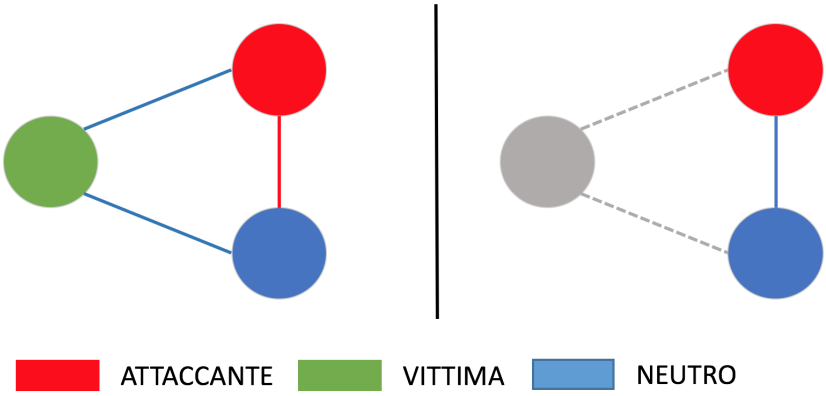

Figura 4.4: Attacco di tipo *slandering*. Il nodo vittima subisce l'estromissione dalla rete di reputazione a causa della diffusione di false informazioni provenienti dal nodo attaccante.

messaggio m, inoltrato mediante la primitiva di comunicazione $MPI\_Isend(m_{ij})$ viene sostituito con un messaggio generato ad hoc, contenente la falsa informazione. I nodi destinatari j non saranno in grado di individuare tale falsificazione.

Il secondo attacco prevede l'adozione di un comportamento egoistico, da parte di uno o più nodi della rete, nei confronti della comunità . Quando il tipo di comportamento malizioso è individuato dallo RMS, il nodo parassita si ripresenta alla comunità sotto altra identità , ripulendo di fatto la propria reputazione. Al fine di simulare tale attacco, al progettista è delegato l'onere di individuare l'insieme dei nodi parassita e la soglia di reputazione sotto la quale possono essere individuati come tali. La simulazione di questo scenario implica il coinvolgimento del simulatore ad entrambi i livelli di astrazione. Per capirne il funzionamento si osservi la figura 4.5.

Al livello più alto un nodo parassita che protrae il suo comportamento dannoso, viene presto rilevato ed estromesso dalla rete. Al livello di astrazione più basso il processo A che simula tale comportamento, nell'istante in cui i servizi ricevuti dalla comunità si riducono drasticamente, avvia il processo di cambio di identità . Esso infatti invierà una richiesta di duplicazione al processo *controller* C introdotto precedentemente. Quest'ultimo, si occuperà di individuare un processo E dai processi inattivi, trasferendo le caratteristiche del processo A su di esso, mutandone lo stato. Il processo E, che fino a quell'istante era assolutamente invisibile al livello di reputazione, riceverà la lista dei vicini e il tipo di comportamento del processo A. Il processo E da questo istante muterà il suo stato in attivo. Dal *round* successivo tale nodo verrà riconosciuto

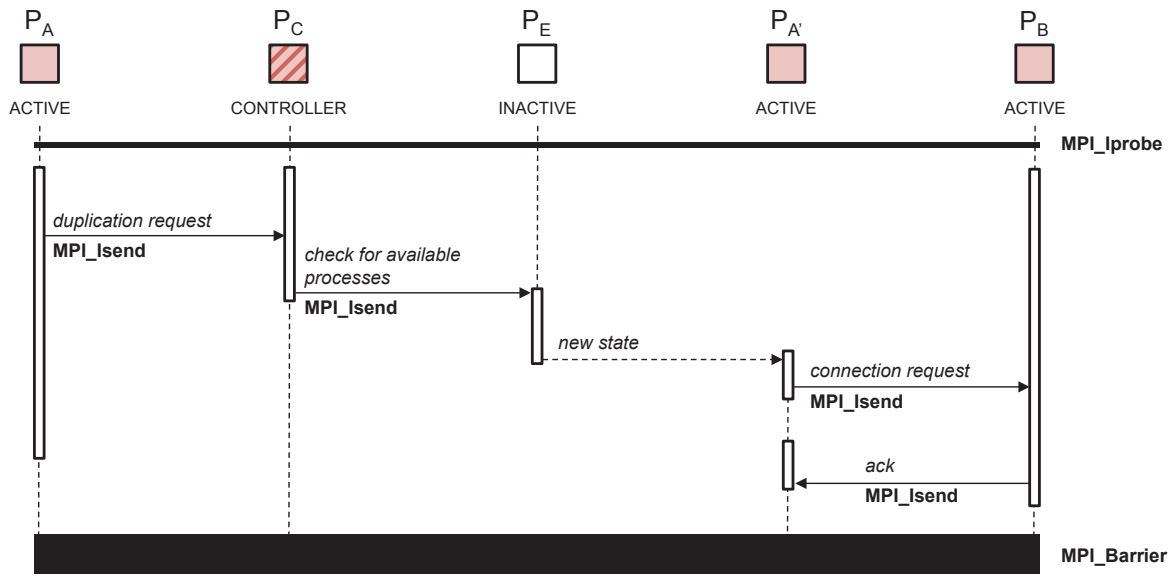

Figura 4.5: Diagramma di comunicazione al livello di simulazione del processo di duplicazione

nella rete in particolare si ripresenterà ai nodi vicini del processo A, ricevendo un nuovo valore di reputazione con i relativi benefici. Il nodo A avrà di fatto assunto una nuova identità A', e potrà continuare ad adottare il proprio comportamento dannoso. Il progettista dello RMS avrà interesse ad osservare la velocità impiegata nel rilevare tali comportamenti dannosi ed eventualmente adottare strategie per limitare il fenomeno sopra discusso.

Capitolo 5

Risultati Sperimentali

In questo capitolo si vogliono mostrare le potenzialità espresse dal *framework* di simulazione e la sua versatilità che lo rende adatto ad essere eseguito sia su sistemi con limitate risorse computazionali che su cluster ad alte prestazioni. Nella prima parte del capitolo si presentano tutti i passi per la sua configurazione, specificando tutte le componenti software e hardware utilizzate. Nella seconda parte verranno mostrati nel dettaglio gli scenari di attacco, ed i relativi *settings*, e in che modo è possibile effettuare delle valutazioni sul RMS, infine verranno discussi i risultati sperimentali ottenuti.

5.1 Setup ambiente, VM e cluster

Il *framework* di simulazione proposto mostra grandi potenzialità in termini di versatilità, esso infatti si adatta facilmente al quantitativo di risorse computazionali a disposizione. Risulta necessario predisporre almeno un calcolatore linux con un'implementazione di MPI. Per i risultati presentati in seguito sono state utilizzate le seguenti versioni:

- Ubuntu 15.04
- MPICH-3.1

Come mostra la figura 5.1, l'intuizione di fare corrispondere un nodo del sistema di gestione della reputazione ad un processo, incrementa sensibilmente la potenzialità dell'esecuzione parallela, qualora l'hardware lo permetta. Sebbene sia possibile condurre esperimenti su un solo calcolatore, per apprezzare significativamente le potenzialità, occorre configurare un cluster di

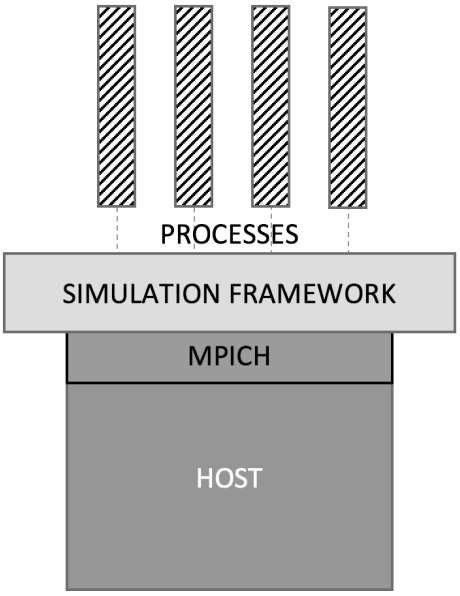

Figura 5.1: Struttura framework di simulazione su singolo host

calcolatori. L'interfaccia di comunicazione MPI, di fatto abilita la comunicazione tra processi, a prescindere dal fatto che essi siano eseguiti sullo stesso hardware. Affinché una simulazione sia eseguibile su più calcolatori, è necessario dotarle dello stesso corredo software, di un account utente con stesso nome utente e password, e di una *working-directory* condivisa con *Network File System* (NFS), o di una sua esatta replica.
L'interfaccia di comunicazione MPICH, oltre a fornire le primitive di comunicazione tra processi, permette la loro esecuzione parallela su un numero arbitrario di calcolatori attraverso il protocollo SSH. Come è noto, SSH è un protocollo di rete, il quale permette di stabilire sessioni remote cifrate tramite interfaccia a riga di comando. Risulta opportuno pertanto dotare ognuna del pacchetto software openSSH-server. Affinché si stabilisca una sessione remota è necessario un passo di autenticazione, che di norma avviene inserendo interattivamente la *password* dell'host remoto. Come è prevedibile, una fase di autenticazione tramite password non è plausibile nello scenario sopra descritto; risulterebbe impraticabile fornire la password manualmente ogni qualvolta si voglia eseguire in maniera distribuita un programma MPI. A tal fine, risulta necessario utilizzare un'autenticazione a chiave pubblica.

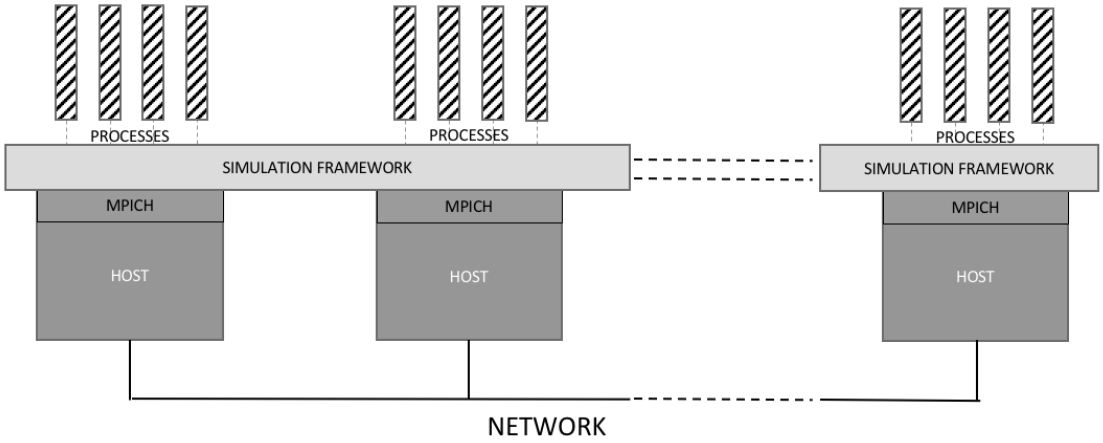

Figura 5.2: Struttura framework di simulazione su un *pool* di host

Poiché l'esecuzione di un'applicazione è comunque avviata da una dei calcolatori del *pool* (figura 5.2), è conveniente identificarne uno ed utilizzarlo a tal fine, con lo scopo di ridurre al minimo il numero di mutue autenticazioni. Riassumendo, una volta stabilito il calcolatore attraverso il quale lanciare l'applicazione, è necessario che gli altri calcolatori del *pool*, siano in grado di autenticare i suoi messaggi. Il primo passo, nel processo di configurazione del *cluster*, consiste nella generazione da parte del *host* 1 di una coppia di chiavi pubblica e privata. Compiuto questo passo, non rimane che distribuire la chiave pubblica all'intero *pool*, che a questo punto sarà in grado di autenticare i messaggi di *host* 1, firmati da quest'ultimo con la rispettiva chiave privata. Il processo appena descritto si concretizza attraverso il generatore di chiavi RSA di SSH e l'invio della chiave pubblica tramite scp, la quale verrà appesa nel file delle chiavi autorizzate degli host remoti; il tutto richiede l'inserimento della password del host remoto una volta per ogni host del *pool*. La fase di configurazione dell'ambiente di sviluppo termina con l'inserimento di un file di configurazione nel *host 1*, il quale contiene una lista di host, ovvero l'indirizzo IP di ognuna dei calcolatori del *pool*, e il corrispettivo numero di processi da affidare in esecuzione a tale calcolatore. Nella tabella in basso è possibile osservare la struttura esemplificativa di un generico *hostfile*:

Nome host:	Numero di processi
localhost:	2
mpi-2:	4
192.168.23.3:	2
...:	...
mpi-x:	y

La prima riga identifica *host* 1, al quale vengono affidati due processi, e di seguito il nome host dei calcolatori presenti nel pool e i corrispettivi processi.

Configurazione ambiente di test

La valutazione sperimentale del sistema proposto in questo lavoro di tesi è stata realizzata utilizzando un *cluster* di risorse hardware ad alte prestazioni. Il corredo hardware disponibile è formato da N. 2 lame HP ProLiant DL560 Gen8 ognuna delle quali dotate di:

- N. 4 socket;
- N. 8 core per socket;
- Processore Intel(R) Xeon(R) CPU E5-4627 v2 @ 3.30 GHz;
- 128 GB RAM.

La sua gestione è a carico del software di virtualizzazione vSphere web client 5.5.0. Si è deciso di predisporre l'installazione delle seguenti macchine virtuali:

- n. 64 macchine virtuali (32 per lama);
- n 2 CPU per Virtual Machine (VM);
- n. 2 GB per VM;
- username: mpi@mpi-x con x compreso tra [1,64].

Dallo schema in figura 5.3 è possibile osservare la disposizione delle macchine virtuali e la loro connessione in rete. La macchina virtuale mpi-1 è dotata di doppia interfaccia di rete, una che la espone su Internet, e l'altra all'interno della rete virtuale privata. Le rimanenti

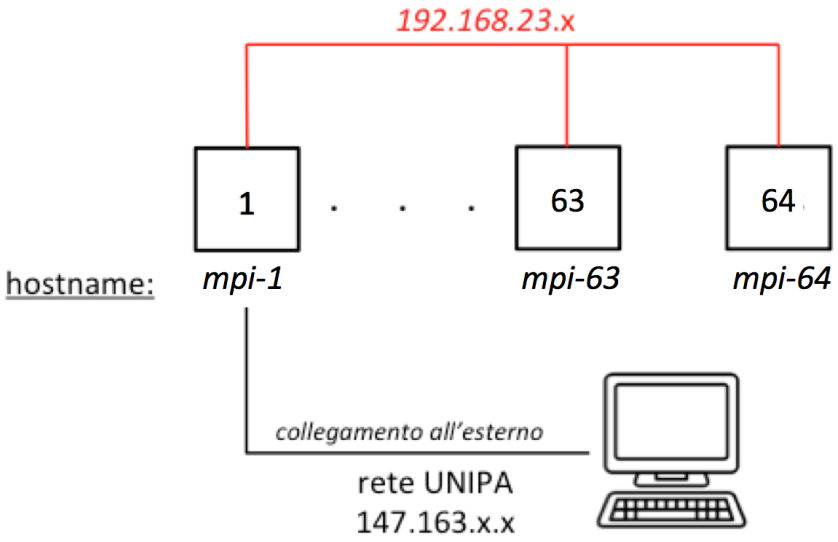

Figura 5.3: Cluster di 64 macchine virtuali su cui testare il framework di simulazione

macchine sono dotate di una singola interfaccia di rete che le espone sulla rete privata del tipo 192.168.23.xxx. L'assegnazione degli indirizzi IP sulla rete privata è delegata ad un server DHCP installato sul host 1 e dotato di un file di configurazione che associa l'indirizzo MAC delle schede di rete virtuali sempre al medesimo indirizzo IP nel range visto precedentemente, in tal modo la macchina il cui nome host è del tipo mpi-1 ha indirizzo IP 192.168.23.1 e così via.
Per la generazione delle macchine si è provveduto all'utilizzo della funzionalità di clonazione messa a disposizione dal software vSphere.

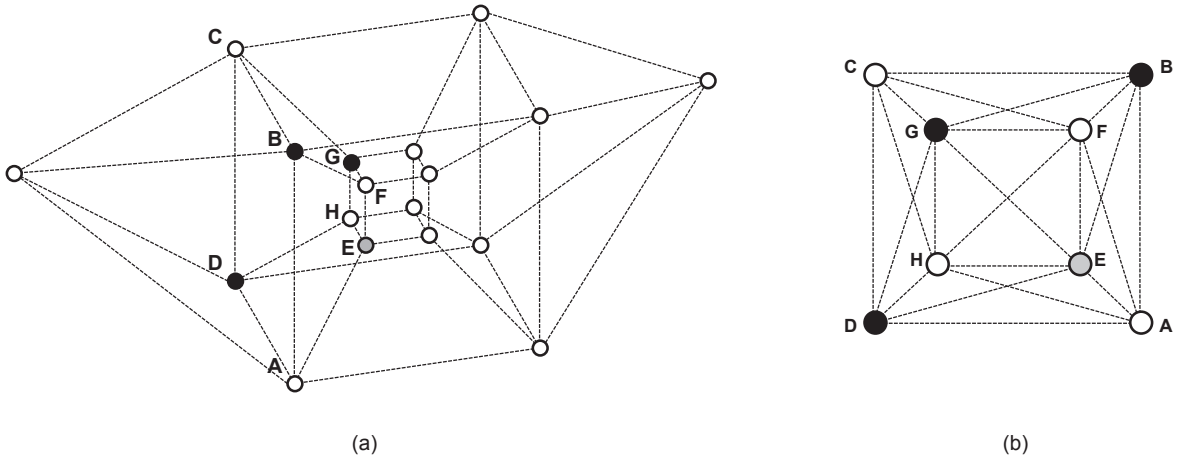

Figura 5.4: Topologia di rete individuata per la conduzione degli esperimenti (a). Sezione laterale della topologia (b).

5.2 Risultati sperimentali

Per verificare l'efficacia del RMS individuato quale caso di studio nel capitolo 3, sono state condotte un insieme di prove sperimentali. In particolare si è voluto osservare la reazione del sistema di gestione della reputazione ad alcuni degli attacchi più comuni studiati in letteratura: lo *slandering* e il *whitewashing attack*. Attraverso i risulti ottenuti emerge il ruolo giocato dal meccanismo di incentivi e dell'algoritmo di stima e di diffusione della reputazione. Ogni esperimento sarà accompagnato da alcune considerazioni utili alla sua valutazione e dal tipo di intervento che si potrebbe realizzare al fine di migliorarlo.

5.2.1 Scenario di simulazione

La simulazione è stata condotta sulla rete raffigurata in figura 5.4. Il comportamento dei nodi viene definito specificandone un valore compreso tra 0 e 1. Tali valori contraddistinguono il grado di cooperazione di un singolo nodo. Un valore prossimo allo zero individua i cosiddetti *free rider*, ovvero quei nodi caratterizzati da un comportamento egoistico e che cercano di trarre profitto dalla condivisione di risorse.

Si è scelto di impostare un valore diverso da zero affinché il comportamento sia più verosimile, infatti i *free rider* occasionalmente decidono di cooperare per trarre in inganno il resto della comunità. Il valore iniziale di reputazione, per ogni nuovo inserimento, è stato posto ad

1. Tale scelta, sebbene discutibile dal punto di vista della progettazione di un RMS, in quanto incrementa le possibilità di successo di un attacco di tipo *whitewashing*, risulta utile per mostrare la capacità del simulatore di rilevare tali comportamenti. La soglia di reputazione τ oltre la quale un nodo è considerato cooperativo è di 0.6. Il valore di β, ovvero il coefficiente utilizzato per integrare le informazioni di reputazione provenienti dalla comunità, è di 0.1. I valori appena descritti sono riassunti nella seguente tabella:

Parametro	**Funzione**	**Valore**
N	Numero di nodi simulati	18
Coop_node	Grado di cooperazione di un generico nodo	1.0
Freerider_node	Grado di cooperazione di un nodo dannoso	0.1
τ	Valore di reputazione oltre il quale considerare le informazioni attendibili	0.6
β	Coefficiente di fusione delle informazioni	0.6

5.2.2 Valutazione dell'attacco di *Slandering*

Questo test mostra la possibilità di simulazione dell'attacco di tipo *slandering*. Per la conduzione di tale attacco ci si riferisce allo scenario mostrato nella figura 5.5, che rappresenta un sottoinsieme della topologia completa. Quello che è possibile vedere in figura è solo uno degli scenari realizzabili potenzialmente illimitati: il progettista può infatti prevedere qualsiasi scenario e il coinvolgimento di un arbitrario numeri di nodi. È possibile osservare la presenza di tre diverse tipologie di nodi: i nodi in bianco N, i nodi in nero M e un nodo in grigio V. La prima categoria di nodi è contraddistinta da un comportamento di tipo collaborativo nel rilascio di risorse nella rete, e neutro nella trasmissione delle informazioni di reputazione. I nodi di tipo M, invece, pur mantenendo un profilo di collaborazione nel rilascio di risorse, alterano volontariamente i valore di reputazione di un nodo vittima V, contraddistinto da un alto grado di cooperazione.

Il simulatore, come già anticipato precedentemente, durante il processo di simulazione colleziona tutte le informazioni riguardo la percentuale di risorse percepite e valori di reputazione percepiti da ogni singolo nodo rispetto ai suoi vicini. Al termine della simulazione, il *framework* avrà collezionato i valori di reputazione e la percentuale di risorse ottenute per tutti gli agenti coinvolti nella simulazione istante per istante. Il confronto fra la reputazione stimata per un

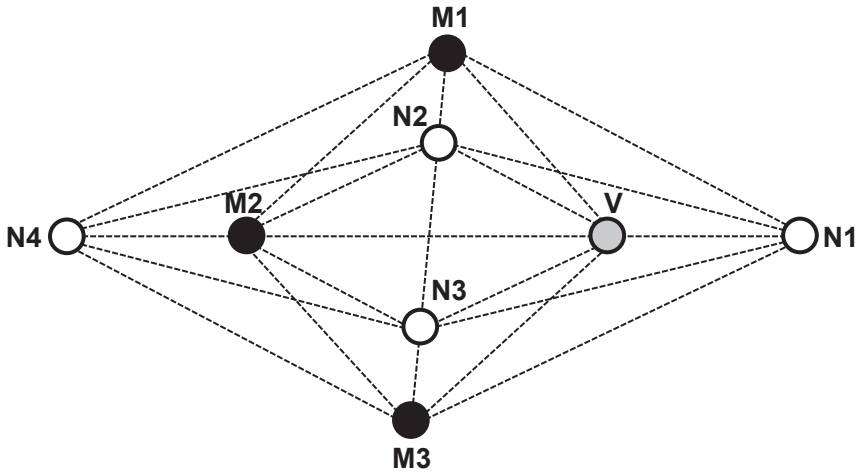

Figura 5.5: Scenario per la conduzione dell'attacco di tipo slandering. In bianco i nodi cooperativi, in nero i nodi attaccanti, in grigio il nodo vittima.

dato agente rispetto al suo grado di cooperazione, permette di valutare l'accuratezza media e istantanea di un RMS. Al fine di incrementarne l'efficacia e la resistenza del RMS in esame si può intervenire sulla modifica di alcuni parametri illustrati di seguito.
Si è deciso di osservare l'evoluzione della simulazione dal punto di vista di uno dei nodi coinvolti nello scenario, ovvero il nodo N1.

Ognuno dei nodi adotta il proprio comportamento a partire dall'istante di inizio della simulazione, che si svolge in 500 round. Nella figura 5.6 è mostrato l'andamento temporale della reputazione del nodo vittima V dal punto di osservazione del nodo N1 e della percentuale di risorse ricevute. Il grafico evidenzia che gli effetti dell'attacco non sono immediati, grazie al contributo dei nodi di tipo N e alle interazioni dirette con il nodo vittima. Intorno al centesimo step però il livello di reputazione percepito dall'osservatore inizia a decrementare, in conseguenza dell'azione diffamatoria congiunta dei nodi di tipo M, i quali diffondono valori di falsa reputazione pari a 0. Come è possibile osservare dal grafico in basso le conseguenze per il nodo vittima sono disastrose: esso vede diminuire l'utilità percepita in maniera proporzionale al valore di reputazione percepito dalla comunità per effetto del meccanismo di incentivi. La valutazione del RMS in oggetto può essere fatta tenendo in considerazione le curve sopra mostrate: in questo caso è auspicabile diminuire gli effetti della diffusione delle false informazioni per cui il progettista potrebbe ridurre il valore del coefficiente β attraverso il quale si aggiorna il valore

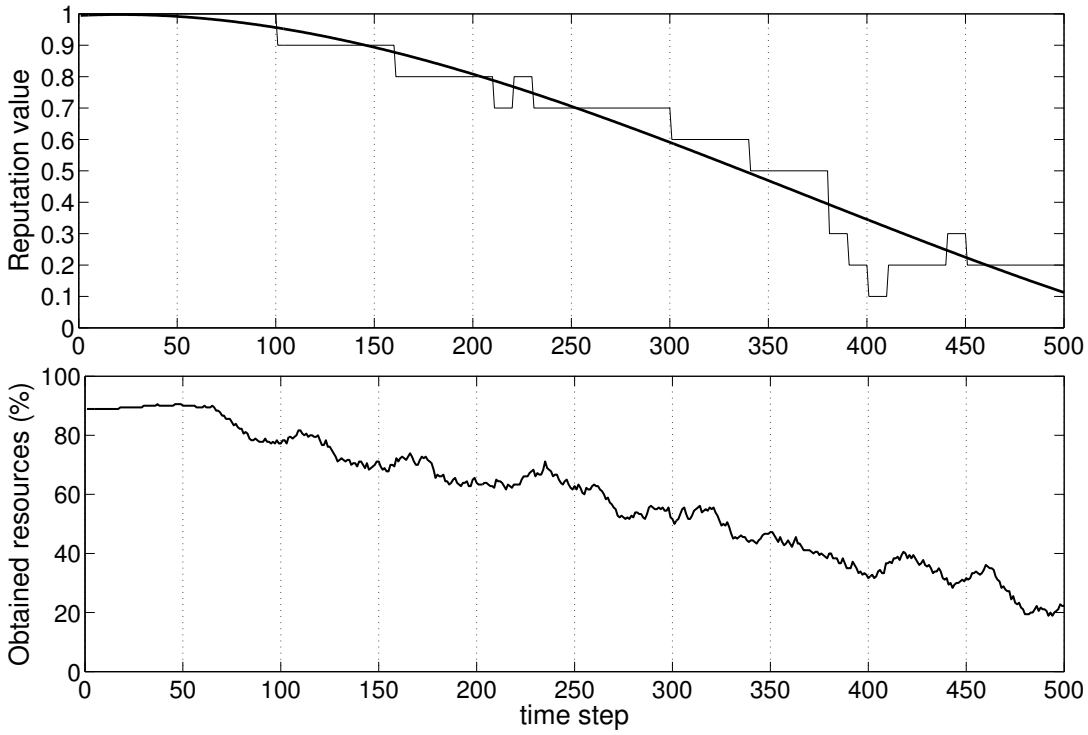

Figura 5.6: Andamento al variare degli step del valore di reputazione del nodo vittima dello *slandering* e della percentuale di risorse da esso ricevute.

di reputazione, dando in definitiva maggiore peso alle informazioni dirette. Si potrebbe inoltre variare il valore di soglia τ in modo da tenere in considerazione solo i nodi con reputazione maggiore. Esso potrebbe infine ridefinire i meccanismi di diffusione e di fusione delle informazioni. In generale, qualora sia previsto dal particolare scenario applicativo la considerevole presenza di questo tipo di attacco, il progettista potrebbe scegliere un RMS il cui setting di parametri permetta la rappresentazione di curve che mantengano una migliore accuratezza per un numero di step maggiore.

5.2.3 Valutazione dell'attacco di Whitewashing

Una analisi simile alla precedente può essere condotta per l'attacco di tipo *whitewashing*. Questo tipo di attacco vede la presenza di due categorie di nodi: i nodi cooperativi, contraddistinti in figura 5.7 con il nome di B e D, e i nodi *free rider*, i nodi A e A'. Alla prima categoria viene assegnata un grado di cooperazione pari ad 1. Alla rimanente viene assegnato un valore di 0.1. La simulazione qui mostrata consiste nell'esecuzione di 500 step temporali, in cui il sistema di

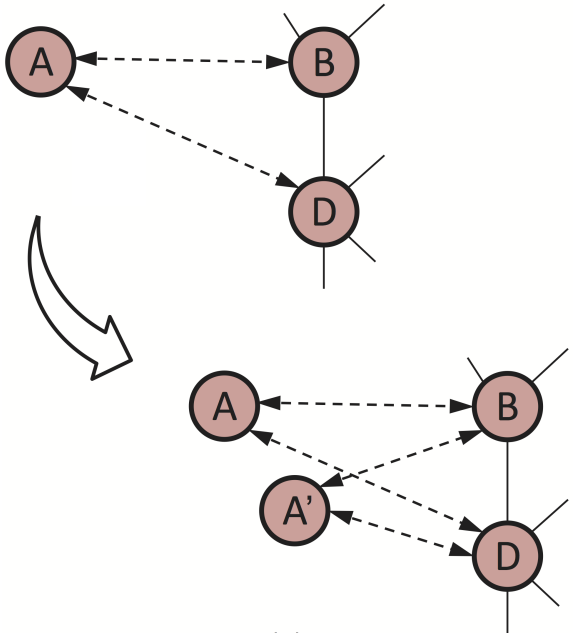

Figura 5.7: Simulazione attacco *whitewashing*. Il nodo A, una volta individuato dal RMS come dannoso, duplica la propria identità ripresentandosi al sistema con un nuovo pseudonimo, A'.

reputazione oggetto dell'analisi, provvede al riconoscimento e alla penalizzazione dei comportamenti dannosi. I nodi che subiscono tale penalizzazione, provvedono alla loro duplicazione, rientrando nel sistema sotto falso pseudonimo, A' in figura. Per farsi un'idea del tipo di reazione del RMS in analisi è possibile osservare i dati raccolti in figura 5.8.

Il grafico in alto mostra il valore di reputazione del nodo *free rider* dal punto di vista di osservazione di uno dei nodi vicini. In particolare la prima curva mostra l'andamento della reputazione del nodo A dal punto di vista di B. A causa del comportamento egoistico del nodo A, il valore di reputazione tende velocemente ad una diminuzione per effetto del sistema di reputazione. Conseguentemente, per effetto del meccanismo di incentivi, anche l'utilità percepita dal nodo A scende proporzionalmente.
Il simulatore a questo punto innesca il meccanismo di duplicazione che permette al nodo A di ripresentarsi al sistema di reputazione sotto falso pseudonimo, A'. Il nuovo nodo riceve in un primo momento i benefici del nuovo arrivato, avendo un nuovo valore di reputazione, per cui può continuare a manifestare il proprio comportamento egoistico. La seconda curva in entrambi i grafici consente l'apprezzamento del fenomeno appena descritto.

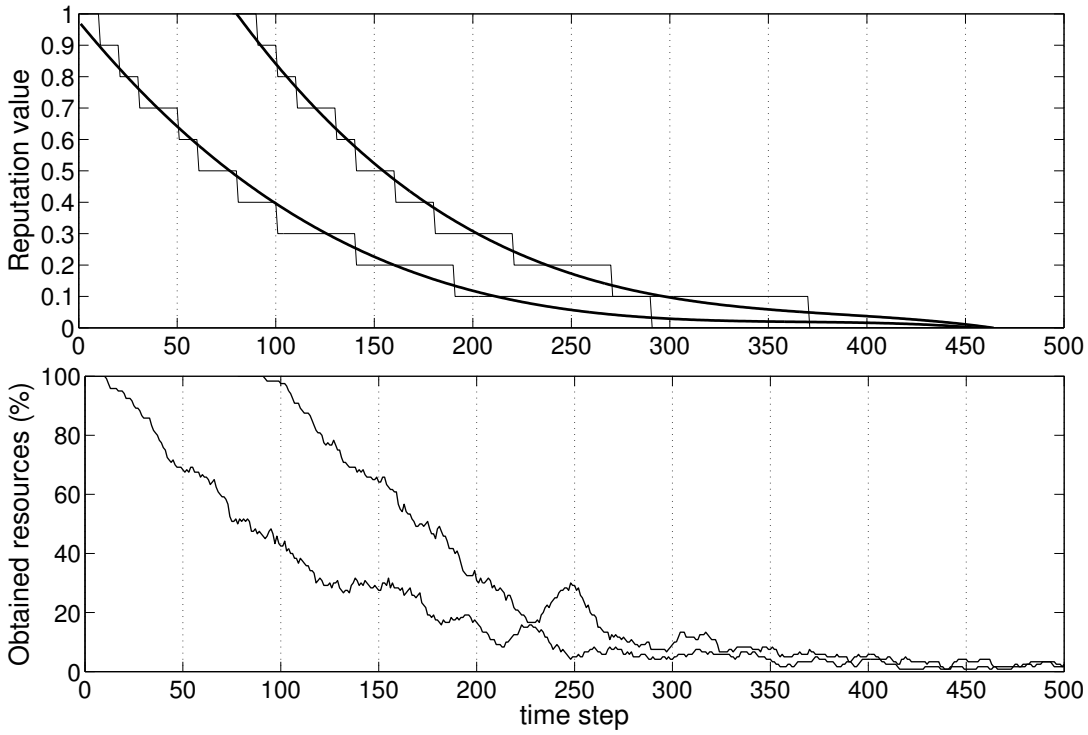

Figura 5.8: Andamento al variare degli step del valore di reputazione dei nodi *free rider* nel *whitewashing* e della percentuale di risorse da esso ricevute.

La valutazione del RMS può essere eseguita in termini della velocità di individuazione del comportamento malevolo. Confrontando i grafici della variazione istantanea del valore di reputazione e della percentuale di risorse ottenute con quelli di altri RMS, al fine di scongiurare tale tipologia di attacco, è necessario che sia la reputazione che l'utilità percepita decrescano il più velocemente possibile. È inoltre opportuno scegliere un RMS in cui le curve dei nodi A e A' siano temporalmente il più distanti possibili: in un tale sistema infatti i tempi necessari al nodo malevolo per ripresentarsi sono più lunghi per cui potrebbe esserne un ottimo deterrente alla conduzione di questo attacco.

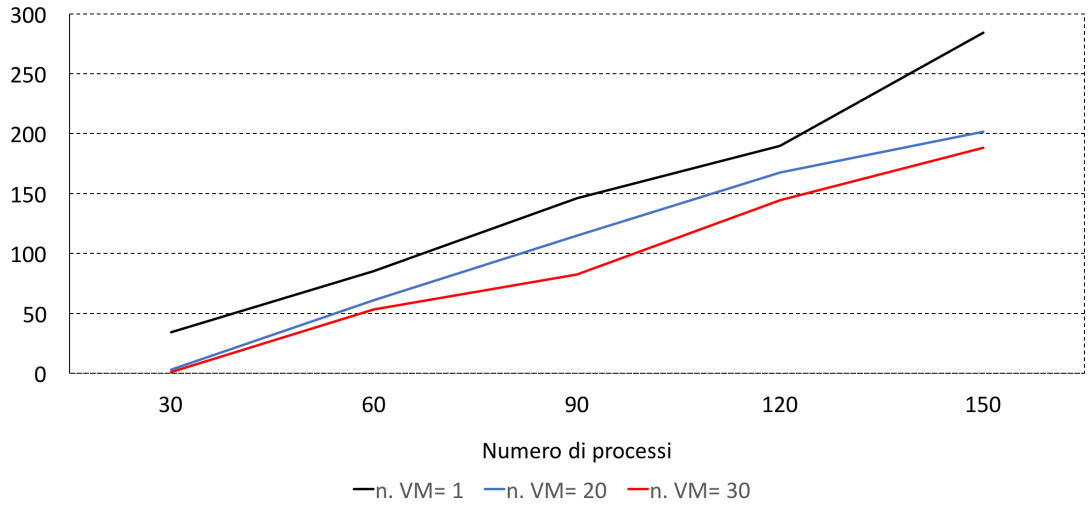

Figura 5.9: Tempo di esecuzione espresso in secondi della simulazione (500 round), al variare del numero di macchine virtuali componenti il cluster di risorse computazionali, e del numero di agenti.

5.2.4 Test di scalabilità del framework

Come precedentemente anticipato, la configurazione del *cluster* ha reso possibile l'incremento delle prestazioni del *framework* di simulazione in termini di scalabilità. Nella figura 5.9 è possibile osservare i tempi di esecuzione al variare del numero di agenti coinvolti e del numero di macchine virtuali componenti il *cluster*. Sono presenti tre curve, ognuna delle quali descrive il tempo (in secondi) impiegato per l'esecuzione di una simulazione di 500 *round* al variare del numero di agenti coinvolti, ognuno associato ad un processo. Il grafico dimostra la convenienza nell'utilizzo di un maggiore quantitativo di macchine virtuali a parità di processi in esecuzione. La curva in nero in particolare mostra i tempi di esecuzione utilizzando un solo calcolatore. L'esecuzione risulta più lenta di almeno un ordine di grandezza rispetto all'utilizzo di 20 e 30 VM già a partire dal più basso numero di agenti.
L'incremento del numero di macchine virtuali giustifica la contrazione dei tempi di esecuzione. L'aumento del numero di processori disponibili aumenta il grado di parallelismo offerto, diminuendo il numero di *context switches* totali per l'esecuzione parallela. Nella figura 5.10 è possibile osservare tale fenomeno. Il grafico mostra il numero di *context switches* totali durante l'esecuzione di una simulazione di 30 processi al variare del numero di macchine virtuali. In particolare risulta massimo quando l'esecuzione implica l'utilizzo di una sola macchina virtuale, mentre decresce all'aumentare del numero di macchine virtuali utilizzate. Risulta minimo

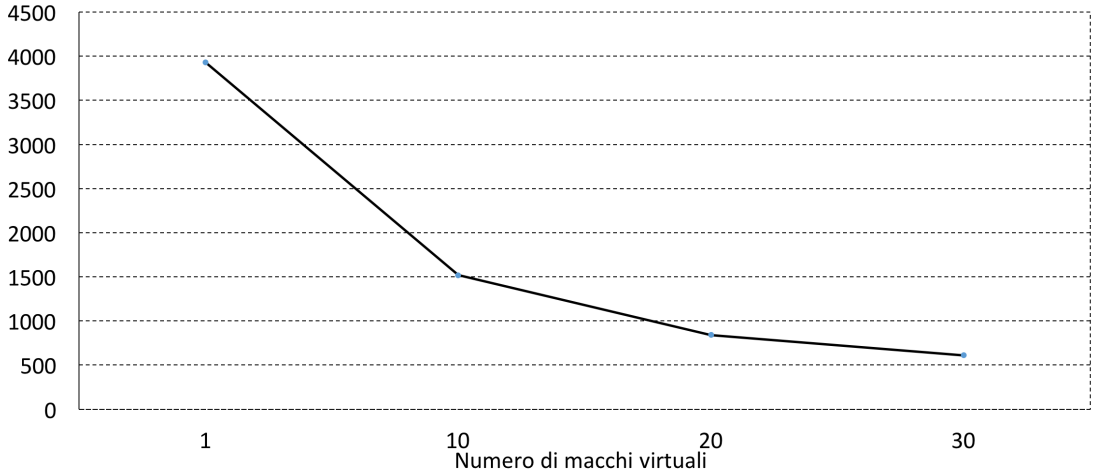

Figura 5.10: Numero di *context switches* necessari per l'esecuzione della simulazione di 30 agenti in 500 round al variare del numero di macchine virtuali.

quando il *mapping* processo-VM è pari ad 1:1, ovvero per il cluster di 30 macchine virtuali.

Conclusioni

In questo lavoro di tesi è stato proposto un ambiente di simulazione parallelo al fine di supportare la valutazione di modelli di gestione della reputazione in ambienti distribuiti.
Sono state descritte le motivazioni che spingono all'adozione dei sistemi di reputazione e le caratteristiche che spesso li accomunano.
Sono stati analizzati alcuni dei RMS, evidenziando le principali vulnerabilità a cui risultano suscettibili, e i simulatori presentati in letteratura allo scopo di supportarne la valutazione sperimentale.
La progettazione del simulatore ha impiegato i concetti teorici propri degli algoritmi distribuiti per modellare i RMS. L'adozione di tale modello si è riflessa nell'impiego del paradigma di comunicazione tra processi attraverso la specifica MPI, un protocollo di comunicazione a passaggio di messaggi.
Il modello architetturale del *framework* di simulazione, sfrutta l'intuizione di associare ogni agente del sistema di reputazione ad un processo. Tale caratteristica lo rende incline alla possibilità di esecuzioni sia su architetture hardware di piccole dimensioni che su cluster di calcolatori collegati in rete, rendendo possibili simulazioni su larga scala.
Il simulatore è risultato un valido strumento per la valutazione di un generico RMS. È stata svolta la valutazione sperimentale di uno specifico RMS, individuato quale caso di studio, valutandone la resistenza all'attacco di tipo *slandering* e all'attacco di tipo *whitewashing*, due degli attacchi più comuni che affliggono i sistemi di reputazione. Si è mostrato in che modo il progettista possa ridefinire il modello di gestione della reputazione, intervenendo sulle funzionalità offerte dal simulatore.
Il framework, infine, ha mostrato buone capacità in termini di scalabilità, dimostrando la possibilità di essere eseguito su un numero generico di calcolatori, riducendo i tempi di simulazione.
Una più approfondita analisi della scalabilità del *framework* per simulazioni su larga scala, in termini di occupazione della memoria, complessità computazionale e di comunicazione, potreb-

be essere effettuata come lavoro di ricerca futuro, sfruttando tecniche avanzate per l'allocazione efficiente dei processi [9, 10]. Inoltre si potrebbe adottare il simulatore per la modellazione delle interazioni all'interno dei social networks, ad esempio Twitter [7, 8].

Elenco delle figure

Bibliografia

[1] Chandrasekaran, P., Esfandiari, B.: Toward a testbed for evaluating computational trust models: experiments and analysis. Journal of Trust Management 2(1), 1–27 (2015)

[2] Crapanzano, C., Milazzo, F., De Paola, A., Lo Re, G.: Reputation management for distributed service-oriented architectures. In: Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010 Fourth IEEE International Conference on. pp. 160–165. IEEE (2010)

[3] De Paola, A., Tamburo, A.: Reputation management in distributed systems. In: Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on. pp. 666–670. IEEE (2008)

[4] Farmer, F.R., Glass, B.: Costruire sistemi per la reputazione Web. Tecniche Nuove (2011)

[5] Fouliras, P.: A novel reputation-based model for e-commerce. Operational research 13(1), 113–138 (2013)

[6] Fullam, K.K., Klos, T.B., Muller, G., Sabater, J., Schlosser, A., Topol, Z., Barber, K.S., Rosenschein, J.S., Vercouter, L., Voss, M.: A specification of the agent reputation and trust (art) testbed: experimentation and competition for trust in agent societies. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. pp. 512–518. ACM (2005)

[7] Gaglio, S., Lo Re, G., Morana, M.: Real-time detection of Twitter social events from the user's perspective. In: Proc. of the 2015 IEEE Int. Conf. on Communications (ICC). pp. 1207–1212 (2015)

[8] Gaglio, S., Lo Re, G., Morana, M.: A framework for real-time Twitter data analysis. Computer Communications 73, Part B, 236 – 242 (2016)

[9] Genco, A., Lo Re, G.: A recognize-and-accuse policy to speed up distributed processes. In: Proceedings of the thirteenth annual ACM symp. on Principles of distributed computing. p. 386. ACM (1994)

[10] Genco, A., Lo Re, G.: The egoistic approach to parallel process migration into heterogeneous workstation network. Journal of Systems Architecture 42(4), 267–278 (1996)

[11] Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the mpi message passing interface standard. Parallel computing 22(6), 789–828 (1996)

[12] Hoffman, K., Zage, D., Nita-Rotaru, C.: A survey of attack and defense techniques for reputation systems. ACM Computing Surveys (CSUR) 42(1), 1 (2009)

[13] Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: Proceedings of the 12th international conference on World Wide Web. pp. 640–651. ACM (2003)

[14] Kerr, R., Cohen, R.: Smart cheaters do prosper: defeating trust and reputation systems. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2. pp. 993–1000. International Foundation for Autonomous Agents and Multiagent Systems (2009)

[15] Lalomia, A., Lo Re, G., Ortolani, M.: A hybrid framework for soft real-time wsn simulation. In: Proc. of the 13th IEEE/ACM Int. Symp. on Distributed Simulation and Real Time Applications, 2009 (DS-RT '09). pp. 201–207 (2009)

[16] Lee, J., Zhang, J., Huang, Z., Lin, K.J.: Context-based reputation management for service composition and reconfiguration. In: Commerce and Enterprise Computing (CEC), 2012 IEEE 14th International Conference on. pp. 57–61. IEEE (2012)

[17] Lynch, N.A.: Distributed algorithms. Morgan Kaufmann (1996)

[18] Marti, S., Garcia-Molina, H.: Taxonomy of trust: Categorizing p2p reputation systems. Computer Networks 50(4), 472–484 (2006)

[19] Russell, S.J., Norvig, P.: Intelligenza artificiale. Un approccio moderno, vol. 1. Pearson Italia Spa (2005)

[20] Santoro, N.: Design and analysis of distributed algorithms, vol. 56. John Wiley & Sons (2006)

[21] Saphir, W.: A survey of mpi implementations. Lawrence Berkeley National Laboratory, University of California, Berkeley, CA, Nov. 6 (1997)

[22] Sharma, S.V., Gopalakrishnan, G., Kirby, R.M.: A survey of mpi related debuggers and tools. Salt Lake City, UT 84112 (2007)

[23] Weng, J., Shen, Z., Miao, C., Goh, A., Leung, C.: Credibility: How agents can handle unfair third-party testimonies in computational trust models. Knowledge and Data Engineering, IEEE Transactions on 22(9), 1286–1298 (2010)

[24] Yu, H., Liu, S., Kot, A.C., Miao, C., Leung, C.: Dynamic witness selection for trustworthy distributed cooperative sensing in cognitive radio networks. In: Communication Technology (ICCT), 2011 IEEE 13th International Conference on. pp. 1–6. IEEE (2011)

[25] Yu, H., Shen, Z., Leung, C., Miao, C., Lesser, V.R.: A survey of multi-agent trust management systems. Access, IEEE 1, 35–50 (2013)