UNIVERSITÀ DEGLI STUDI DI UDINE

Dipartimento di Matematica e Informatica

Corso di Laurea Magistrale in Informatica

Tesi di Laurea

SVILUPPO DI UN SISTEMA DISTRIBUITO MODULARE E SCALABILE PER LA PUBBLICAZIONE DI SERVIZI VO

Relatore:
Dott. Ivan Scagnetto

Laureando:
Francesco Cepparo

Correlatore:
Dott. Marco Molinaro

ANNO ACCADEMICO 2014-2015

Indice

Elenco delle figure **v**

1 Caso d'uso **3**

2 Stato dell'arte **5**
2.1 Definizione interfaccia e implementazione RPC 5
2.1.1 WSDL 6
2.1.2 Facebook/Apache Thrift 7
2.1.3 Apache Avro 7
2.1.4 Cisco/Apache Etch 8
2.1.5 Google Protocol Buffers 9
2.1.6 Eclipse Franca IDL 10
2.1.7 JSON-WSP 10
2.1.8 ZeroC Ice 11
2.2 Implementazione di microservizi ed EJB 12
2.2.1 Stateful session beans 14
2.2.2 Stateless session beans 14
2.2.3 Singleton session beans 14
2.2.4 Message-driven beans 15
2.3 Framework per sistemi distribuiti 15
2.3.1 Akka 16
2.3.2 Jini 16
2.3.3 Jade 17
2.3.4 Jolie 18
2.4 Automazione dello sviluppo 18
2.4.1 Ant 18
2.4.2 Maven 19
2.4.3 Gradle 20
2.5 Middleware per publish/subscribe 21
2.5.1 UDDI 21
2.5.2 XMPP 22

2.5.3 MQTT . . . 23
2.5.4 AMQP . . . 24
2.5.5 JMS . . . 24
2.5.6 JGroups . . . 25
2.6 Framework per il logging . . . 26
2.6.1 Log4j . . . 26
2.6.2 Logback . . . 26
2.6.3 Log4j 2 . . . 27
2.6.4 tinylog . . . 28
2.6.5 Jakarta Commons Logging . . . 29
2.6.6 Simple Logging Facade for Java . . . 29
2.7 Alta disponibilità e load balancing . . . 30
2.7.1 HAProxy . . . 30
2.7.2 nginx . . . 31
2.8 Sistemi di virtualizzazione . . . 32
2.8.1 Xen . . . 32
2.8.2 KVM . . . 33
2.8.3 VMware Workstation . . . 33
2.8.4 VirtualBox . . . 34
2.8.5 Vagrant . . . 35
2.8.6 Docker . . . 35

3 Tecnologie utilizzate 37

4 Architettura del sistema 41
4.1 Il Reverse Proxy . . . 41
4.2 L'Interfaccia . . . 44
4.3 Il Message Broker . . . 44
4.4 L'Access Data Layer . . . 45
4.5 Il Logging Server . . . 45
4.6 Il Config Repository . . . 45

5 Test e valutazione del sistema 47

6 Conclusioni e sviluppi futuri 49

Bibliografia 51

Elenco delle figure

4.1 L'architettura del sistema 42
4.2 Il diagramma di sequenza in versione semplificata 43
4.3 Il diagramma di sequenza 43

Introduzione

L'Osservatorio Virtuale (VO) è un insieme di tecnologie, standard, e software per la pubblicazione e l'accesso a dati astronomici. L'International Virtual Observatory Alliance (IVOA) è un'organizzazione che stabilisce gli standard di interoperabilità che rendono possibile l'esistenza del VO [1]. Il centro Italiano Archivi Astronomici IA2 ospitato presso l'INAF-OATs (Istituto Nazionale di AstroFisica - Osservatorio Astronomico di Trieste) possiede un software per la pubblicazione di servizi legati al VO chiamato VO-Dance, che soffre di problemi legati alla scalabilità e alla complessità del codice, che lo rende difficilmente manutenibile in rapporto all'evoluzione degli standard VO. Il progetto di tesi nasce in questo contesto, per fornire al centro dati IA2 un software con un'architettura alternativa - facilmente manutenibile, distribuita, modulare e scalabile - per la pubblicazione di servizi VO.

Per dimostrare la fattibilità del sistema, nell'ambito della tesi sono stati sviluppati due servizi di tipo Simple Cone Search, uno standard che definisce un protocollo per recuperare informazioni da un catalogo di dati astronomici a partire da una posizione sulla sfera celeste e da una distanza angolare, parametri che permettono di definire un cono nel cielo [2]. I servizi che supportano questo standard, se interrogati, devono fornire in risposta una lista di sorgenti astronomiche le cui posizioni giacciano all'interno di tale cono. Tale risposta deve essere codificata nel formato VOTable, formato standard del VO basato sullo eXtensible Markup Language (XML) e atto allo scambio di dati rappresentati in forma tabulare [3].

Capitolo 1

Caso d'uso

Con questo lavoro di tesi si vuole fornire un sistema per la pubblicazione di servizi VO, temporaneamente denominato VOBall. In particolare, si vogliono supportare richieste HTTP GET parametriche e fornire in risposta delle tabelle in formato VOTable.

Il sistema che si vuole sostituire, denominato VO-Dance, presenta diversi problemi, che con questo lavoro di tesi si vogliono affrontare e risolvere.

Innanzitutto, VO-Dance consiste in un'unica applicazione web monolitica, che si occupa di pubblicare servizi VO multipli. Tuttavia, in caso di bug o crash dell'applicazione, tutti i servizi vengono a mancare. Il nuovo sistema è invece stato concepito come modulare, sia dal lato dell'interfaccia, sia dal lato dell'accesso ai dati, il che previene in modo naturale il problema di cui sopra.

Un ulteriore problema di VO-Dance consiste nella limitata flessibilità dell'applicazione nel supportare nuovi tipi di servizi e protocolli VO. Infatti, con l'evoluzione di tali servizi e protocolli non è più possibile uniformarne l'implementazione utilizzando le Java Reflection, come avviene in VO-Dance, ma è necessario scrivere un nuovo modulo per aggiungerne il supporto. Nel caso di VOBall, grazie alla modularità e alla minimalità del codice questo è molto semplice, mentre, nel caso di VO-Dance, anche data la complessità del codice, aggiungere il supporto ad un nuovo protocollo sarebbe uno sforzo non da poco.

Un altro vincolo posto dall'architettura monolitica di VO-Dance consiste

nell'utilizzo obbligatorio di Java anche per l'accesso alle basi di dati. Con VOBall invece, è possibile supportare in modo trasparente diversi linguaggi di programmazione e diverse modalità di accesso ai dati.

Inoltre, VO-Dance presenta problemi di scalabilità, in quanto l'applicazione è legata ad un'unica macchina e non permette di distribuire il calcolo su più nodi. Con VOBall invece si possono distribuire con facilità sia le interfacce sia i moduli di accesso ai dati, permettendo una totale distribuzione del carico, oltre a garantire alta disponibilità.

Infine, VO-Dance possiede un'interfaccia di amministrazione scritta in Python, di difficile installazione e configurazione, e la configurazione dei servizi va inserita in un database. Con VOBall invece la configurazione dei servizi avviene tramite semplici file di proprietà in formato testuale. Un'eventuale interfaccia di amministrazione viene considerata come eventuale sviluppo futuro.

Capitolo 2

Stato dell'arte

In questo capitolo si passa in rassegna lo stato dell'arte sulle tecnologie per l'implementazione di servizi distribuiti. Molte informazioni sono tratte dai siti ufficiali delle rispettive tecnologie, ma anche da Wikipedia inglese [1] e dalle rispettive voci. Questo step del lavoro di tesi è stato necessario per individuare le tecnologie più adatte allo sviluppo del prototipo che descriveremo in seguito, nel capitolo 4.

2.1 Definizione interfaccia e implementazione RPC

La definizione di interfacce formali si effettua tramite i cosiddetti framework IDL (Interface Definition Language), che a partire dalla definizione formale di un'interfaccia forniscono un'implementazione per Remote Procedure Call (RPC) / Remote Method Invocation (RMI) e un abbozzo di client e server (stub e skeleton) specifici per tale interfaccia, in diversi linguaggi di programmazione. Ogni framework IDL specifica il proprio linguaggio formale per la definizione di interfacce (DSL, Domain Specific Language). Di framework IDL ce ne sono a bizzeffe, e di seguito si passeranno in rassegna i maggiori esponen-

[1] http://en.wikipedia.org

ti della categoria analizzandone le caratteristiche e valutandone così l'idoneità all'adozione per il lavoro di tesi.

2.1.1 WSDL

Il Web Services Description Language è un IDL basato su XML e atto alla descrizione delle funzionalità offerte dai servizi web. Per WSDL si intende anche il descrittore specifico di un singolo web service. In particolare, un file WSDL fornisce una descrizione di come il servizio possa essere chiamato, quali parametri esso si aspetti, e quali strutture dati esso ritorni, rendendolo simile ad una type signature tipica dei linguaggi di programmazione. Il WSDL descrive i servizi come insiemi di endpoint, chiamati anche porte. Le definizioni astratte di porte e messaggi sono raggruppate in una sezione separata rispetto alla sezione in cui vengono descritti i loro usi concreti (istanze). Una porta viene definita associando un indirizzo di rete a un binding, e un insieme di porte definisce un servizio. I messaggi sono descrizioni astratte dei dati che vengono scambiati, mentre i tipi di porte sono insiemi astratti di operazioni supportate. Il protocollo concreto e le specifiche riguardanti il formato dei dati per un particolare tipo di porta costituisce un binding riutilizzabile, in cui le operazioni e i messaggi sono legati ad un protocollo di rete e ad un messaggio concreto. In questo modo il WSDL descrive l'interfaccia pubblica al web service.

Il WSDL è spesso utilizzato in combinazione con il SOAP (in origine acronimo di Simple Object Access Protocol) e uno schema XML per fornire servizi web su internet. Un client che si colleghi ad un web service può leggere il file WSDL per determinare quali operazioni sono disponibili sul server. Qualunque tipo di dato speciale utilizzato è incluso nel file WSDL sotto forma di schema XML. Il client può a quel punto utilizzare SOAP per chiamare una qualsiasi delle operazioni elencate nel file WSDL utilizzando ad esempio l'XML attraverso l'HTTP.

La versione attuale della specifica WSDL è la 2.0. La versione 1.2 è stata rinominata in 2.0 a causa delle differenze sostanziali rispetto al WSDL 1.1: accettando binding a tutti i metodi HTTP (ovvero non solo GET e POST, come accade nella versione 1.1), la versione 2.0 offre un migliore supporto ai web services RESTful. La versione 2.0 è diventata una raccomandazione del W3C nel 2007 [4].

2.1.2 Facebook/Apache Thrift

Thrift è un IDL e un protocollo di comunicazione binario che viene utilizzato per definire e creare servizi in diversi linguaggi di programmazione. Viene utilizzato come framework RPC, ed è stato originariamente sviluppato da Facebook [2] per lo sviluppo scalabile e cross-language di servizi. Combina uno stack software con un motore di generazione del codice per costruire servizi che funzionino in modo efficiente e in modo trasparente rispetto al linguaggio di programmazione utilizzato. Thrift include uno stack completo per la creazione di client e server. I layer di trasporto e di protocollo fannno parte della libreria utilizzata a runtime, ed è quindi possibile cambiarne l'implementazione senza ricompilare il proprio codice. Thrift permette una serializzazione e comunicazione a basso overhead rispetto ad alternative come SOAP grazie all'utilizzo del formato binario. Thrift supporta numerosi linguaggi di programmazione: C, C++, C#, D, Delphi, Erlang, Go, Haxe, Haskell, Java, JavaScript, Node.js, OCaml, Perl, PHP, Python, Ruby, e Smalltalk [5]. Thrift è attualmente un progetto open source della Apache Software Foundation [3].

2.1.3 Apache Avro

Avro è un framework per l'RPC e per la serializzazione, sviluppato come parte del progetto Apache Hadoop [4]. Utilizza JSON (JavaScript Object No-

[2] http://www.facebook.com
[3] http://www.apache.org
[4] https://hadoop.apache.org

tation) per definire i tipi di dato e i protocolli, e serializza i dati in un formato binario compatto. Funziona in modo simile a Thrift, ma non richiede un programma per la generazione del codice quando si cambia lo schema descrittore del servizio, a meno che non si desideri utilizzare un linguaggio con tipizzazione statica. I linguaggi supportati da Avro sono C, C++, C#, Java, Perl, Python, Ruby e PHP [6]. Oltre a JSON, Avro include un supporto sperimentale per un suo proprio linguaggio IDL per la descrizione di interfacce [7], il cui formato è progettato in modo da facilitarne l'adozione da parte degli utenti che hanno familiarità con linguaggi di programmazione e IDL più tradizionali, con una sintassi simile a quella del C/C++ e dei Protocol Buffers (descritti nella sezione 2.1.5).

2.1.4 Cisco/Apache Etch

Apache Etch è un framework multipiattaforma, indipendente dal linguaggio di programmazione, per la costruzione e l'utilizzo di servizi di rete [8]. Etch include un linguaggio per la descrizione dei servizi di rete, un compilatore, e librerie per il binding con diversi linguaggi di programmazione. Etch è anche indipendente dal livello di trasporto, il che permette l'utilizzo di diversi tipi di trasporto a seconda delle necessità e delle circostanze. Lo scopo di Etch è quello di rendere semplice la definizione di piccoli servizi specializzati, rendendoli di facile accesso, permettendo di combinarli e di metterli in produzione. Etch è nato dalla necessità di avere una descrizione concisa e formale dello scambio di messaggi tra un client e un server, con alcuni requisiti di base: alte performance e scalabilità, supportare una comunicazione in tempo reale a una o due vie, supportare client e server scritti in linguaggi di programmazione diversi, supportare client e server in esecuzione in contesti diversi, supportare la creazione di nuovi binding a nuovi linguaggi di programmazione e strati di trasporto, essere una libreria leggera e veloce ma anche flessibile abbastanza da rispondere ai requisiti di cui sopra, e infine essere facile da usare per gli

sviluppatori che implementano o consumano i servizi. I linguaggi al momento supportati sono Java, C#, e C. Il supporto al C++ è in beta, mentre in stato alpha esiste il supporto a Go, JavaScript e Python.

2.1.5 Google Protocol Buffers

I Protocol Buffers sono un modo per serializzare dati strutturati. Sono utili quando si sviluppano programmi che necessitano di comunicare l'uno con l'altro o immagazzinare dati. La realizzazione prevede un IDL che descriva la struttura dei dati e un programma che generi del codice sorgente a partire da tale descrizione, il quale a sua volta si occupi di generare o effettuare il parsing dello stream di byte che rappresentano i dati strutturati.

Google ha sviluppato i Protocol Buffers per uso interno, implementando il supporto a C++, Java e Python e rendendo poi open source tali implementazioni. Tuttavia esiste il supporto anche a molti altri linguaggi, tramite implementazioni non ufficiali di terze parti [9]. Gli obiettivi nella progettazione dei Protocol Buffers miravano ad enfatizzare la semplicità e le performance. In particolare, sono stati progettati per essere più piccoli e veloci del codice XML. I Protocol Buffers sono ampiamente usati da Google per immagazzinare e scambiare qualsiasi tipo di informazione strutturata. I Protocol Buffers vengono usati come base per un sistema personalizzato di RPC, e il loro protocollo è molto simile a quello di Apache Thrift, anche se non includono uno stack completo per l'RPC.

I Protocol Buffers funzionano nel modo seguente: lo sviluppatore definisce strutture dati (chiamate messaggi) e servizi in un file di definizione .proto, e lo compila con protoc. La compilazione genera del codice che può essere invocato dal mittente o dal destinatario di queste strutture dati. Il formato di serializzazione dei Protocol Buffers è compatto e compatibile sia in avanti che all'indietro, ma non è auto-descrittivo, ovvero non c'è modo di recuperare i nomi, il significato, o i tipi di dato dei campi senza utilizzare una specifica

esterna, e non è definito alcun modo per includere o per riferirsi ad una specifica esterna dall'interno di un file Protocol Buffer. L'implementazione ufficiale include un formato di serializzazione in ASCII, che è auto-descrittivo ma che perde la compatibilità in avanti e indietro, e che perciò è adatto solamente per il debugging.

2.1.6 Eclipse Franca IDL

Franca IDL è un linguaggio per la descrizione di interfacce formalmente definito. Fa parte del framework Franca, che si occupa anch'esso della definizione e della trasformazione di interfacce software. Franca applica tecniche di trasformazione dei modelli per l'interoperabilità con diversi linguaggi di definizione di interfacce.

La versione iniziale di Franca è stata sviluppata dal consorzio GENIVI [5] nel 2011 come linguaggio comune per una piattaforma IVI (In-Vehicle Infotainment). La prima versione pubblica di Franca è stata rilasciata nel marzo del 2012 con la licenza Eclipse Public Licence. Nel 2013, Franca è stato proposto come progetto ufficiale della fondazione Eclipse [10], ed è principalmente sviluppato dalla compagnia tedesca Itemis.

2.1.7 JSON-WSP

JSON-WSP (JavaScript Object Notation Web-Service Protocol) è un protocollo per i web service che usa JSON per la descizione dei servizi, per le richieste, e per le risposte. È molto ispirato a JSON-RPC, ma nasce per la mancanza in quest'ultimo di una specifica di descrizione dei servizi. Il formato di descrizione ha lo scopo di definire i tipi e i metodi utilizzati da un dato servizio. Descrive anche relazioni tra tipi diversi (come ad esempio i tipi annidati) e definisce quali tipi ci si aspetti come argomenti per un metodo e quali tipi l'utente possa aspettarsi di ricevere come valori di ritorno. Infine, il formato

[5] http://www.genivi.org

permette di aggiungere documentazione ai livelli di servizio, metodo, parametro, e valori di ritorno. La comunicazione tra un client e un server JSON-WSP viene effettuata tramite richieste e risposte HTTP POST e con oggetti JSON come dati, con content-type application/json.

JSON-WSP consiste di quattro specifiche di oggetti JSON: description, request, response, e fault. L'oggetto description descrive metodi, parametri, tipi, e tipi di ritorno, e supporta la documentazione utente sui livelli di servizio, metodo, e parametro. L'oggetto request contiene informazioni riguardanti quale metodo dev'essere invocato e tutti gli argomenti per la chiamata al metodo. Gli argomenti nella richiesta devono rientrare nella definizione dei parametri del metodo stesso, come descritti nella sezione corrispondente. L'oggetto response contiene il risultato di un'invocazione ad un metodo del servizio. Il tipo di ritorno deve rispettare il tipo di ritorno definito nella sezione descrittiva corrispondente. Infine, l'oggetto fault specifica se è occorso qualche errore lato client oppure lato server. A seconda del framework in uso sul server, è possibile risalire al file e al numero di riga in cui è avvenuto l'errore.

2.1.8 ZeroC Ice

Ice, acronimo di Internet Communications Engine, è una piattaforma middleware orientata agli oggetti che fornisce funzionalità di RPC, grid computing e publish/subscribe. Sviluppata da ZeroC [6], la piattaforma è rilasciata con licenza doppia, una proprietaria e una open source (la GNU General Public Licence, versione 2) [11]. Supporta i linguaggi C++, C#, Visual Basic, Java, Objective-C, PHP, Python, e Ruby [12].

Ice è stato influenzato nel suo design dalla Common Object Request Broker Architecture (CORBA), infatti è stato sviluppato da diversi sviluppatori CORBA. Secondo ZeroC tuttavia, Ice è più leggero e meno complesso di CORBA

[6] https://zeroc.com

poiché progettato da un piccolo gruppo di sviluppatori con esperienza aventi una visione complessiva del progetto [13].

Ice è un insieme di diverse componenti, che includono invocazione remota di oggetti, replicazione, grid-computing, failover, bilanciamento del carico, attraversamento di firewall e servizi publish/subscribe. Per accedere a questi servizi, le applicazioni si appoggiano ad una libreria stub generata da un linguaggio IDL chiamato slice.

2.2 Implementazione di microservizi ed EJB

Le EJB (Enterprise JavaBeans) sono una tecnologia per la costruzione modulare di software enterprise, e consistono in una Java API. In particolare, le EJB sono un componente software lato server che incapsula la business logic di un'applicazione. La specifica delle EJB [14] è un sottoinsieme delle specifiche Java EE [15].

La specifica per le EJB intende fornire una modalità standard per implementare software lato server tipicamente presente nelle applicazioni enterprise. In particolare, intende fornire una API per problematiche comuni, come persistenza, sicurezza, e integrità, in modo che i programmatori non debbano reinventare l'acqua calda ogni volta e possano concentrarsi sulle parti specifiche del software enterprise da sviluppare.

La specifica delle EJB definisce quali facilities deve fornire un application server. Secondo la specifica EJB un application server deve supportare le seguenti feature: transaction processing, integrazione con i servizi di persistenza offerti dalle Java Persistance API (JPA), controllo della concorrenza, programmazione guidata da eventi tramite utilizzo di Java Message Service e Java EE Connector Architecture, invocazione asincrona dei metodi, scheduling dei job, servizi di naming e di directory (Java Naming and Directory Interface), interprocess communication tramite RMI-IIOP e web services, sicurezza (Java Cryptography Extension e Java Authentication and Authorization Service), e

deployment di componenti software. Inoltre essa definisce i ruoli dei container EJB e di come effettuare il deployment dell'EJB in un container. Tuttavia, l'attuale specifica EJB (versione 3.2) non descrive come un application server debba fornire la persistenza, delegando questo compito alle specifiche JPA. Ciò che la EJB specifica invece, riguarda il modo in cui il codice dell'applicazione possa essere facilmente integrato con i servizi di persistenza offerti dall'application server.

L'ultima revisione delle EJB, la versione 3, è frutto di un radicale cambiamento rispetto alle versioni precedenti, e segue un nuovo paradigma, che consiste nell'utilizzare Plain Java Objects e nel fornire il supporto alle dependency injections per semplificare la configurazione e l'integrazione di sistemi eterogenei. Gavin King, il creatore di Hibernate, partecipò alla creazione delle EJB 3.0 e ne è un noto sostenitore. Molte caratteristiche originarie di Hibernate sono state incorporate nelle JPA, che rimpiazzano le entity beans nelle EJB 3. Le EJB 3 si basano pesantemente sull'uso di annotazioni (introdotte in Java 5.0) e convenzioni per permettere la scrittura di codice molto più conciso. In pratica, le EJB 3 sono molto più leggere rispetto alle versioni precedenti e forniscono una API quasi totalmente nuova, tanto da rispecchiare solo lontanamente le versioni precedenti.

Un container EJB può contenere due tipi principali di beans: i beans di sessione e i message-driven beans. I session beans si suddividono a loro volta in stateful, stateless e singleton, e si può accedere ad essi localmente, tramite la stessa JVM, oppure da remoto, in quest'ultimo caso sia tramite interfaccia che in modo diretto. Tutti i tipi di session beans supportano l'esecuzione asincrona, così come i message-driven beans, i quali tuttavia la supportano tramite scambio messaggi.

2.2.1 Stateful session beans

Le stateful session beans sono oggetti che posseggono uno stato, ovvero tengono traccia di quale client sono in comunicazione attraverso una sessione, e l'accesso all'istanza del bean è strettamente limitata a un solo client alla volta. Se si tenta di accedere al bean in modo concorrente, il container serializza le richieste, almeno finché non si raggiunge un tempo di timeout specificato tramite annotazione, nel qual caso il container potrà generare un'eccezione. Lo stato delle stateful session beans può essere mantenuto tramite persistenza in modo automatico da parte del container, in modo da poter liberare memoria se il client non accede al bean per un certo periodo di tempo. Questo comportamento è supportato in modo esplicito dalle stateful session beans.

2.2.2 Stateless session beans

Le stateless session beans sono oggetti che non hanno uno stato associato ad esse. Tuttavia, l'accesso ad una singola istanza è anche in questo caso limitato ad un client alla volta, e l'accesso concorrente è proibito. Se questo avviene, il container dirotta le richieste concorrenti ad istanze diverse. Questo rende le stateless session beans automaticamente thread-safe. Variabili d'istanza possono essere usate al momento della chiamata da parte di un client, ma non è garantito che il loro valore sia preservato tra chiamate diverse. Le istanze delle stateless session beans sono tipicamente mantenute in un pool. Nel momento in cui termina la comunicazione fra un bean ed un client, è possibile che nel caso in cui arrivi una seconda richiesta, anche da parte di un client diverso, sia la stessa istanza a rispondere. La mancanza dello stato rende le stateless session beans meno intensive computazionalmente rispetto alla loro controparte stateful.

2.2.3 Singleton session beans

Le singleton session beans sono oggetti con uno stato globale all'interno della JVM. L'accesso concorrente all'unica istanza può essere controllato dal

container (CMC, Container Managed Concurrency) oppure dal bean stesso (BMC, Bean Managed Concurrency). La concorrenza CMC può essere controllata tramite l'annotazione @Lock, che specifica se un lock dev'essere utilizzato durante la chiamata ad un metodo. Infine, le singleton session beans possono richiedere in modo specifico di essere istanziate al momento dell'avvio del container, tramite l'utilizzo dell'annotazione @Startup.

2.2.4 Message-driven beans

Le message-driven beans (MDB) sono oggetti la cui esecuzione è composta da messaggi al posto di chiamate a funzioni. Le message-driven beans sono utilizzate per fornire un'astrazione ad alto livello per la specifica JMS (Java Message Service) che opera a basso livello. Sono stati aggiunti alle EJB per permettere il processing basato sugli eventi. A differenza delle session bean, una message-driven bean non possiede un aggancio lato client (locale, remoto, o senza interfaccia), e per questo i client non possono effettuare il look up di un'istanza MDB. Tutto ciò che fa una message-driven bean è ascoltare se arrivano messaggi, per esempio su una coda JMS, e processarli automaticamente. Le specifiche Java EE richiedono solo il supporto a JMS, ma le message-driven bean posssono supportare anche altri protocolli di scambio messaggi. Tali protocolli possono essere sincroni o asincroni. Per questo motivo, la differenza principale con le stateful session beans consiste nello scambio messaggi piuttosto che chiamate a metodi.

2.3 Framework per sistemi distribuiti

In questa sezione si passano in rassegna i framework per sistemi distribuiti, ovvero librerie che permettano di facilitare lo sviluppo di sistemi distribuiti grazie all'astrazione fornita.

2.3.1 Akka

Akka è un toolkit open source per semplificare la costruzione di applicazioni concorrenti e distribuite basate sulla JVM. Akka supporta diversi modelli di programmazione per la concorrenza, ma enfatizza una concorrenza basata sugli attori e prende ispirazione dal linguaggio di programmazione Erlang. Esistono binding per i linguaggi Java e Scala. Akka è scritto in Scala, e l'implementazione degli attori di Akka è stata inclusa come parte della libreria standard di Scala [16].

I punti chiave che distinguono le applicazioni basate sugli attori di Akka sono i seguenti. Innanzitutto la concorrenza è basata sullo scambio messaggi ed è asincrona. Tipicamente non vengono condivisi dati modficabili e non vengono usate primitive per la sincronizzazione. Inoltre, il modo con cui gli attori interagiscono è lo stesso, sia che essi risiedano sullo stesso host, o risiedano in host separati, o comunichino direttamente o tramite meccanismi di routing, o siano in esecuzione pochi o molti thread, e così via. Tali dettagli possono essere alterati al momento del deployment attraverso un meccanismo di configurazione, che permette, senza effettuare modifiche al programma, di scalare sia nel caso in cui si faccia uso di pochi server potenti, sia nel caso in cui si voglia utilizzare un numero cospicuo di server.

Akka possiede una struttura modulare, con un modulo principale che implementa gli attori. Gli altri moduli aggiungono caratteristiche come la distribuzione degli attori sulla rete, supporto ai cluster, integrazione con sistemi di terze parti, ma anche supporto ad altri modelli per la concorrenza.

2.3.2 Jini

Jini, noto anche come Apache River, è un architettura di rete per la costruzione di sistemi distribuiti sotto forma di servizi modulari che cooperano fra di loro. Sviluppato originariamente dalla Sun Microsystems, Jini è stato

successivamente rilasciato sotto una licenza open source [17], quella di Apache, ed incluso come progetto nella omonima fondazione.

Introdotto nel 1998, Jini fornisce un'infrastruttura di tipo Service Object Oriented Architecture (SOOA). La localizzazione dei servizi viene fatta attraverso un servizio di lookup. I servizi cercano di contattare un servizio di lookup (LUS), sia tramite unicast (quando il servizio conosce la locazione del lookup service), sia tramite una multicast discovery dinamica. Il servizio di lookup ritorna un oggetto che rappresenta un archivio di servizi che può essere usato dai servizi per registrare loro stessi così che possano essere trovati dai client. I client possono utilizzare il servizio di lookup per recuperare un oggetto che fa da proxy per il servizio. Le chiamate al proxy vengono tradotte in chiamate al servizio. Il proxy effettua la richiesta al servizio, e ritorna il risultato al client. Questa strategia è più flessibile rispetto quella della Java RMI (Remote Method Invocation), poiché quest'ultima richiede al client di conoscere in anticipo la locazione del servizio.

2.3.3 Jade

Quasi acronimo di Java Agent DEvelopment Framework, JADE è un framework implementato nel linguaggio Java. Esso semplifica l'implementazione di sistemi multi-agente attraverso un middleware che rispetta le specifiche FIPA e attraverso un insieme di tool grafici che supportano le fasi di debugging e deployment. Un sistema basato su JADE può essere distribuito attraverso macchine che non necessitano nemmeno lo stesso sistema operativo, mentre la configurazione può essere controllata da una interfaccia grafica remota. La configurazione può anche essere modificata a runtime, spostando agenti da una macchina all'altra, quando richiesto. JADE richiede come minimo la versione 5 del runtime environment di Java.

Oltre all'astrazione degli agenti, JADE fornisce un semplice ma potente modello di esecuzione e composizione di task, comunicazione tra agenti p2p basata

sul paradigma di scambio messaggi asicrono, e un servizio di pagine gialle che supporta meccanismi di publish/subscribe, oltre a molte altre caratteristiche che facilitano lo sviluppo di un sistema distribuito.

2.3.4 Jolie

Jolie è un linguaggio di programmazione open source per sviluppare applicazioni distribuite basate su microservizi. Nel paradigma di programmazione proposto da Jolie, ogni programma è un servizio che può comunicare con altri programmi inviando e ricevendo messaggi attraverso una rete. Jolie supporta un livello di astrazione che permette ai servizi di comunicare attraverso mezzi diversi, dai socket TCP/IP a comunicazioni locali tra processi attraverso la memoria di sistema [18].

Jolie è attualmente supportato da un interprete implementato nel linguaggio Java, e può essere eseguito su più sistemi operativi. Il linguaggio possiede una semantica formale, il che significa che l'esecuzione di programmi Jolie è matematicamente definita. Per questo motivo, Jolie è utilizzato nella ricerca per investigare tecniche basate sui linguaggi di programmazione per lo sviluppo di sistemi distribuiti.

2.4 Automazione dello sviluppo

Automazione dello sviluppo significa automatizzare il processo di compilazione e di pacchettizzazione del codice sorgente, compiti che altrimenti spetterebbero agli sviluppatori stessi. In questa sezione si passano in rassegna i principali tool che permettono la suddetta automazione.

2.4.1 Ant

Apache Ant (Another Neat Tool) [19] è un software concepito per automatizzare il processo di compilazione del software. Originariamente nasce sotto le ali del progetto Tomcat agli inizi dell'anno 2000 come rimpiazzamento del

tool make. Ant è simile a make, ma è implementato utilizzando il linguaggio Java ed è maggiormente adatto alla compilazione dei progetti Java. La più grossa differenza rispetto a make consiste nel fatto che Ant utilizza un file XML per descrivere il processo di compilazione e le sue dipendenze, mentre make utilizza un Makefile. Ant è un progetto Apache open source rilasciato sotto la omonima licenza.

2.4.2 Maven

Maven è un tool per l'automazione della compilazione usato principalmente per progetti Java. Maven affronta due aspetti della compilazione del software: per prima cosa descrive come il software dev'essere compilato, e poi descrive le sue dipendenze. A differenza di tool più datati come Ant, utilizza convenzioni per la procedura di compilazione, in cui vanno specificate solamente le eccezioni. La configurazione del progetto va effettuata in un file XML, in cui viene descritto il progetto, le sue dipendenze, l'ordine di compilazione, le directory sorgenti, ed eventuali plugin. Maven contiene target predefiniti per task ben definiti, come ad esempio la compilazione e la pacchettizzazione. Durante la risoluzione delle dipendenze, Maven scarica automaticamente le librerie e i plugin necessari da uno o più repository, e salva il tutto in una cache locale, che può essere anche aggiornata con artefatti provenienti da progetti locali. Maven può anche essere usato per gestire progetti scritti in C#, Ruby, Scala, e altri linguaggi. [20] [21] Progetti alternativi come Gradle (sezione 2.4.3) e SBT (Scala Build Tool) non utilizzano XML ma mantengono i concetti chiave introdotti da Maven.

I progetti Maven vengono configurati tramite un Project Object Model (POM) file. Generalmente, questa configurazione comprende il nome del progetto e le dipendenze su altri progetti. Si possono anche configurare fasi individuali del processo di compilazione, implementate tramite plugin. Ad esempio, si può configurare il plugin del compilatore in modo che venga utilizzata una

versione specifica di Java, oppure specificare che si vuole ottenere il pacchetto anche se alcune unità di test falliscono. I progetti più grossi dovrebbero essere suddivisi in diversi moduli (sottoprogetti), ognuno con il proprio POM, collegati ad un POM radice attraverso il quale si possono compilare tutti i moduli con un unico comando. I file POM possono anche ereditare la loro configurazione da altri file POM. Di default, tutti i POM ereditano dal POM radice. Il POM radice fornisce la configurazione di default, specificando ad esempio le cartelle sorgente, i plugin, e così via.

La maggior parte delle funzionalità di Maven è fornita dai plugin. Esistono plugin per compilare, testare, e gestire i sorgenti, avviare un web server, generare file di progetto Eclipse, e molti altri ancora. [22] [23] I plugin vengono specificati e configurati nell'apposita sezione del file POM. Alcuni plugin di base sono inclusi di default in ogni progetto, con ragionevoli impostazioni di default. Tuttavia la procedura risulterebbe scomoda se si dovesse ad ogni compilazione invocare ogni goal manualmente. Per questo motivo Maven introduce il concetto di lifecycle [24], ovvero insiemi di goal lanciati in modo sequenziale.

La gestione delle dipendenze è una caratteristica chiave di Maven, e si fonda su un sistema che identifica artefatti come librerie o moduli. Le dipendenze dirette di un progetto vengono specificate nel suo POM, ma vengono valutate e scaricate anche tutte le dipendenze transitive, che vengono poste nel repository locale. Di default viene utilizzato il Maven 2 Central Repository [25] per la ricerca delle librerie, ma si possono configurare repository esterni all'interno del POM.

2.4.3 Gradle

Gradle è un tool per l'automazione dei progetti che si basa sui concetti di Ant e Maven e introduce un linguaggio specifico basato su Groovy invece che sul più tradizionale XML per dichiarare la configurazione di un progetto.

Gradle utilizza un grafo diretto aciclico per determinare l'ordine in cui i task possono essere eseguiti.

Gradle è stato progettato per compilazioni multi-progetto, che possono diventare molto grosse, e supporta compilazioni incrementali determinando in modo intelligente quali parti del codice sono aggiornate, in modo tale da non dover rieseguire task che dipendono su queste parti di codice.

Gradle è compatibile con i progetti Maven e Ant. La compatibilità con Maven deriva dal plugin "java", mentre allo stesso tempo Gradle supporta l'importazione diretta dei build file di Ant come task nativi.

2.5 Middleware per publish/subscribe

Il middleware per publish/subscribe è uno strato intermedio che permette di disaccoppiare due o più componenti che necessitano di comunicare fra di loro. Spesso tale middleware si basa sullo scambio messaggi. In questa sezione si passano in rassegna i sistemi che permettono la comunicazione ed il disaccoppiamento delle varie componenti.

2.5.1 UDDI

Lo UDDI (Universal Description, Discovery and Integration) è un servizio di registry indipendente dalla piattaforma, estensibile, e basato su XML, attraverso il quale le applicazioni possono registrarsi ed essere elencate. UDDI è un'iniziativa sponsorizzata dall'OASIS (Organization for the Advancement of Structured Information Standards) per permettere ai servizi di trovarsi l'uno con l'altro, e definire come questi servizi interagiscono attraverso internet.

Lo UDDI è stato proposto inizialmente come uno standard per i Web Service. È stato progettato per essere interrogato da messaggi SOAP e per fornire accesso ai documenti WSDL che descrivono i binding e i formati richiesti per interagire con i Web Service elencati nella sua directory.

I nodi UDDI sono server che supportano le specifiche UDDI [26] e appartengono ad un UDDI registry, mentre gli UDDI registry sono insiemi di uno o più nodi. Una registrazione UDDI consiste di tre componenti, chiamate pagine bianche, gialle e verdi. Le pagine bianche contengono informazioni riguardo l'azienda che fornisce il servizio. Questo include il nome e una descrizione, potenzialmente scritta in lingue diverse. Usando queste informazioni è possibile trovare un servizio riguardo al quale si conosca già qualche informazione (ad esempio basandosi sul nome del fornitore). Nelle pagine bianche compaiono anche le informazioni di contatto del fornitore del servizio, come ad esempio l'indirizzo e il numero di telefono. Le pagine gialle invece forniscono una classificazione del servizio basandosi su tassonomie standard. Siccome una singola istituzione può fornire un certo numero di servizi, ci possono essere diverse pagine gialle associate ad una singola pagina bianca. Infine, le pagine verdi sono usate per descrivere come effettuare l'accesso ad un Web Service, con informazioni sui binding del servizio. Alcune delle informazioni sono collegate ai Web Service, come l'indirizzo del servizio e i parametri, e i riferimenti alle specifiche delle interfacce. Altre informazioni non sono collegate direttamente al Web Service. Queste includono e-mail, FTP, CORBA, e dettagli telefonici del servizio. Poiché un Web Service può avere binding multipli (come definito nella sua descrizione WSDL), un servizio può avere più pagine verdi (perché ad ogni binding si dovrà accedere in modo diverso).

2.5.2 XMPP

Lo XMPP (Extensible Messaging and Presence Protocol) è un protocollo di comunicazione per middleware orientato ai messaggi e basato sullo XML. Permette lo scambio quasi in real time di dati strutturati ma estensibili tra due o più entità di rete. Il protocollo si chiamava originariamente Jabber, ed è stato sviluppato dalla comunità open source nel 1999 per instant messaging quasi in real time, informazioni sulla presenza, e gestione di una lista di contatti

[27]. Progettato per essere estensibile, il protocollo è stato anche utilizzato per sistemi di publish/subscribe, segnalazioni per il VOIP, video, trasferimento file, gaming, applicazioni Internet of Things e servizi di social networking.

A differenza della maggior parte dei protocolli di messaggistica istantanea, lo XMPP è definito utilizzando uno standard aperto e utilizza un approccio a sistema aperto per lo sviluppo, tramite il quale chiunque può implementare un servizio XMPP e interoperare con le implementazioni di altre organizzazioni. Poiché l'XMPP è un protocollo aperto, le implementazioni possono utilizzare qualunque licenza, anche se molti server, client e librerie che lo implementano sono libere e open source. [28] [29] [30]

L'architettura delle reti XMPP è simile a quella delle email: chiunque può eseguire il proprio server XMPP e non esiste un master server centralizzato. Poiché XMPP è uno standard aperto, chiunque può implementarne le specifiche senza dover pagare royalty, e lo sviluppo non è legato ad una singola azienda. I server XMPP possono essere isolati dalla rete XMPP pubblica, e il protocollo supporta gli standard di sicurezza SASL e TLS.

Il protocollo XMPP ha anche delle debolezze e degli svantaggi. Innanzitutto non supporta la Quality of Service, che va implementata sopra lo strato dell'XMPP. Inoltre XMPP si basa su una comunicazione di tipo testuale e non binaria, il che comporta un maggiore consumo di banda e overhead di rete. Per questo motivo, anche il trasferimento di dati binari è limitato, siccome questi andrebbero prima codificati in base64 prima di poter essere trasmessi in-band. Infine, XMPP non supporta la cifratura end-to-end.

2.5.3 MQTT

MQTT è un protocollo leggero per il public-subscribe che si posiziona sopra il protocollo TCP/IP. È progettato per connessioni con luoghi remoti in cui si richiede del codice minimale e/o in cui la banda è limitata. In realtà, la specifica del protocollo non formalizza e non quantifica le parole "codice minimale"

e "utilizzo di banda limitato" [31]. Per questo l'utilizzabilità del protocollo dipende dal contesto e dall'implementazione. Il modello di publish/subscribe richiede un message broker, ovvero un modulo intermedio che traduce un messaggio dal modello formale del mittente a quello del destinatario. Il broker distribuisce i messaggi ai client interessati basandosi sull'argomento del messaggio.

2.5.4 AMQP

L'AMQP (Advanced Message Queuing Protocol) è un protocollo per middleware orientato ai messaggi [32]. In particolare, l'AMQP è un protocollo binario situato a livello di applicazione, progettato per supportare in modo efficiente una grande varietà di applicazioni di messaggistica e modelli di comunicazione. Lo standard AMQP stabilisce il comportamento dei server e dei client fino al punto da rendere interoperabili implementazioni diverse del protocollo. In precedenza, ad esempio con le JMS, la standardizzazione era stata raggiunta a livello di API, ed era incentrata a permettere l'utilizzo di implementazioni diverse piuttosto che all'interoperabilità fra di esse. L'AMQP invece descrive il formato dei dati che vengono trasmessi, e di conseguenza qualunque tool che possa creare ed interpretare tali messaggi può interoperare con altri tool che facciano lo stesso.

2.5.5 JMS

La Java Message Service è una API orientata allo scambio di messaggi tra due o più client. JMS è parte della Java Platform, Enterprise Edition, ed è definita da una specifica sviluppata sotto il Java Community Process [33]. In pratica, è uno standard di messaggistica che permette a componenti delle applicazioni basate sulla Java Enterprise Edition di creare, inviare, ricevere, e leggere messaggi. Permette alla comunicazione tra diverse compo-

nenti di un'applicazione distribuita di essere debolmente accoppiata, affidabile e asincrona.

La API JMS suppporta due modelli: punto a punto e publish/subscribe. In un sistema di messaggistica punto a punto i messaggi vengono inviati ad un singolo consumatore, che mantiene una coda dei messaggi in ingresso. Questo tipo di messaggistica è costruito sul concetto di code, mittenti, e destinatari. Ogni messaggio è indirizzato ad una coda specifica, e i client destinatari estraggono i messaggi dalla coda che li contiene. È garantito che ogni messaggio venga consegnato e consumato da un destinatario. Le code mantengono tutti i messaggi a loro inviati fino a che i messaggi sono consumati o finché scadono. La coda mantiene i messaggi anche se nessun client è al momento registrato per consumare i messaggi. Il modello publish/subscribe invece supporta la pubblicazione dei messaggi su un particolare topic. I subscriber possono esporre il proprio interesse nel ricevere messaggi di uno specifico topic. In questo modello, il publisher ed il subscriber non si conoscono e non hanno nessuna informazione l'uno dell'altro.

2.5.6 JGroups

JGroups è un toolkit per messaggistica multicast affidabile scritto nel linguaggio Java. Può essere usato per creare cluster i cui nodi possano inviarsi messaggi l'un con l'altro. JGroups aggiunge uno strato di grouping sopra un protocollo di trasporto che mantiene internamente una lista di partecipanti. Questa lista viene utilizzata per rendere un'applicazione cosciente dei listeners, rendere alcune o tutte le trasmissioni affidabili, e permettere trasmissioni ordinate. JGroups può essere usato per creare gruppi di processi i cui membri possano inviarsi messaggi l'un con l'altro. JGroups permette agli sviluppatori di creare applicazioni multicast affidabili, dove l'affidailità è uno dei punti chiave al momento del deployment.

2.6 Framework per il logging

La pratica del logging consiste nel tenere traccia di eventi prestabiliti e nel generarne una raccolta, in modo da avere sotto controllo il funzionamento dell'applicazione e assicurarsi che l'esecuzione proceda correttamente. I log possono anche essere utilizzati per generare delle statistiche riguardanti l'utilizzo dell'applicazione. In questa sezione si passano in rassegna i principali sistemi che aiutino ad effettuare del logging.

2.6.1 Log4j

Log4j è una libreria di logging per Java ed un progetto della Apache Software Foundation, sviluppato da un team dedicato. Fa inoltre parte di un progetto di più ampio spettro, denominato Apache Logging. Log4j permette di effettuare del logging a runtime senza modificare il codice dell'applicazione. Log4j è progettato in modo tale da permettere statement per il logging all'interno del codice senza incorrere in grosse perdite di performance. Il comportamento del logging può essere controllato modificando un file di configurazione, senza toccare il binario dell'applicazione.

Una delle caratteristiche distintive di log4j è la notazione per l'eredità dei logger. Usando una gerarchia di logger è possibile controllare quali statement sono mandati in output con una granularità arbitraria, ma anche con una grande facilità. Questo aiuta a ridurre il volume dell'output e il costo del logging. Il target del log di output può essere un file, un OutputStream, un java.io.Writer, un server log4j remoto, un demone Syslog Unix remoto, e molto altro ancora.

2.6.2 Logback

Logback nasce come successore del popolare progetto log4j, proseguendo dove log4j si è fermato. L'architettura di logback è sufficientemente generica da poter essere applicata a circostanze diverse. Al momento, logback è suddi-

viso in tre moduli: logback-core, logback-classic, e logback-access. Il modulo logback-core pone le fondamenta per gli altri due moduli. Il modulo logback-classic può essere paragonato ad una versione significativamente migliorata di log4j; inoltre, logback-classic implementa nativamente le API SLF4J, così da permettere agli sviluppatori di scambiare logback con altri framework per il logging. Il modulo logback-access si integra con i servlet container come Tomcat e Jetty, per fornire funzionalità di logging per gli accessi HTTP. Se necessario, è possibile scrivere un proprio modulo sopra logback-core.

Ci sono numerose ragioni per preferire logback a log4j [34]. Alcune delle motivazioni principali includono una implementazione più veloce, una estesa collezione di test che dimostrano la solidità di logback anche in condizioni avverse, il supporto nativo a SLF4J, una estensiva documentazione, il supporto a XML o Groovy per il file di configurazione, il ricaricamento automatico dei file di configurazione, una graceful recovery nel momento in cui si verificano errori di I/0, la rimozione automatica dei vecchi log, la compressione automatica dei log archiviati, una "prudent mode" in cui più istanze di FileAppender su multiple JVM possono scrivere tranquillamente sullo stesso file di log, "Lilith" - un viewer dei log e degli eventi di accesso capace di gestire grosse quantità di dati, il processing condizionale dei file di configurazione, un supporto ai filtri molto più dettagliato, ed un SiftingAppender, un appender capace di separare gli eventi di logging a seconda delle sessioni utente.

2.6.3 Log4j 2

Log4j 2 è un'aggiornamento di Log4j 1.x che fornisce significativi miglioramenti riespetto al suo predecessore, e che riesce a fornire anche molti dei miglioramenti propri di Logback, mettendo allo stesso tempo una pezza ad alcuni problemi inerenti all'architettura di Logback. Rispetto a Log4j 1.x, Log4j 2 separa le API dall'implementazione, rendendo chiaro quali classi e metodi gli sviluppatori possono utilizzare, permettendo così di assicurare la compatibilità

in avanti e permettendo al team di Log4j di migliorare l'implementazione in modo sicuro e compatibile. Inoltre, Log4j 2 fornisce logger asincroni di nuova generazione. In scenari multithread, i logger asincroni hanno un throughput maggiore di 18 volte rispetto a Log4j 1.x e Logback, con una latenza inferiore di diversi ordini di grandezza [35]. Log4j 2 supporta API multiple, ed in particolare supporta anche le API di Log4j 1.x, SLF4J, e le Commons Logging API.

Un'altra feature di Log4j 2 consiste nel ricaricamento automatico delle configurazioni. A differenza di Logback inoltre, Log4j 2 permette di farlo senza perdere eventi di logging mentre la riconfigurazione ha luogo. Come Logback invece, Log4j 2 supporta il filtraggio basato su dati contestuali, marker, espressioni regolari, e altri componenti dell'evento di logging. L'architettura di Log4j 2 è basata sui plugin, e come tale non è necessario scrivere codice per creare e configurare le componenti del logger. Log4j riconosce automaticamente i plugin e li usa nel momento in cui questi vengono referenziati nella configurazione. Infine, se nella configurazione vengono referenziate delle proprietà, Log4j le rimpiazza direttamente o le passa ad un componente sottostante che le risolve dinamicamente.

2.6.4 tinylog

Tinylog è un logger minimalistico per Java, ottimizzato per la facilità d'uso: il logger è statico, e non è necessario crearne un'istanza prima di effettuare del logging. Di default tutte le entry di livello info o superiore sono scritte sulla console. Tinylog è configurabile attraverso statement Java, argomenti della JVM, e file di proprietà. Essendo molto piccolo, tinylog è anche molto veloce, anche perché è possibile ottimizzarne l'esecuzione tramite pattern precompilati. Secondo alcuni benchmark, tinylog è fino a quattro volte più veloce di log4j [36]. Un ulteriore vantaggio di tinylog è la scalabilità: essendo thread-safe, tinylog può essere utilizzato in programmi multi-threaded senza la necessità

di utilizzare dei lock. Tutte le entry del log vengono sempre create e scritte in modo atomico. Tinylog è parallelizzabile e trae vantaggio dalla presenza di core multipli nei microprocessori.

2.6.5 Jakarta Commons Logging

Il framework Jakarta Commons Logging (JCL) è un ponte ultra leggero fra diverse implementazioni per il logging. Una libreria che utilizza le API commons-logging può essere utilizzata con una qualsiasi implementazione per il logging a runtime. Commons-logging supporta un discreto numero di implementazioni per il logging, e scrivere nuovi adattatori per nuove implementazioni è un compito ragionevolmente semplice. Anche le applicazioni, oltre alle librerie, possono scegliere di usare commons-logging. Anche se l'indipendenza dall'implementazione del logging non è importante per le applicazioni quanto lo è per le librerie, usare commons-logging permette all'applicazione di cambiare l'implementazione senza ricompilarne il codice.

2.6.6 Simple Logging Facade for Java

SLF4J (Simple Logging Facade for Java) è una semplice astrazione per permettere a diversi logging framework di essere scambiati al momento del deployment su libera scelta dell'utente finale. Abilitare SLF4J comporta l'aggiunta di un'unica dipendenza, ovvero il pacchetto che definisce le API. L'implementazione può essere scambiata dinamicamente senza la necessità di ricompilare l'applicazione, e viene determinata a runtime a seconda del binding inserito nel classpath. La separazione tra frontend e backend permette una maggiore flessibilità e riduce la dipendenza da uno specifico framework per il logging. SLF4J è stato creato come alternativa al framework JCL, che soffre di problematiche legate al class loader. Inoltre, SLF4J risolve in modo pragmatico un problema importante legato alle performance, tramite messaggi di log parametrizzati.

2.7 Alta disponibilità e load balancing

Per poter disporre di un sistema distribuito robusto ed efficiente è necessario valutare come fornire alta disponbilità e load balancing. In questa sezione si passano in rassegna i software che permettono di raggiungere tali obiettivi.

2.7.1 HAProxy

HAProxy è una soluzione libera, affidabile, e molto veloce, atta ad offrire alta disponbilità, bilanciamento del carico, e proxying, per le applicazioni basate su TCP e HTTP. È particolarmente indicato per i siti web con alto traffico, e viene utilizzato in diversi siti tra i più visitati al mondo [37]. Negli anni è diventato lo standard de-facto tra i load balancer open source, ed è attualmente pacchettizzato dalla maggior parte delle distribuzioni Linux, oltre ad essere spesso impiegato in piattaforme cloud.

Per quanto riguarda le performance, HAProxy contiene numerose ottimizzazioni atte a ridurre l'utilizzo della CPU anche con carichi moderati. E con carichi molto alti, quando la CPU viene saturata, si può notare come la maggior parte dei cicli della CPU vengano spesi dal sistema operativo e relative chiamate di sistema. Per questo motivo l'ottimizzazione del sistema operativo è molto importante. In produzione, HAProxy è spesso anche utilizzato come soluzione di emergenza nel momento in cui i load balancer hardware falliscono.

Nei programmi a singolo processo come HAProxy (per minimizzare i context switch e massimizzare le performance), è estremamente importante garantire una completa continuità del servizio e minimizzare la presenza di bug nel codice, ancor più che nei software multi processo, in quanto un piccolo bug può mandare in crash il programma, oppure mandarlo in loop, o bloccarlo. Per questo motivo HAProxy è stato scritto e progettato fin dall'inizio con l'affidabilità come obiettivo. Anche se a breve termine è difficile scrivere un programma affidabile partendo da zero, negli ultimi tredici anni non è stato trovato nessun bug del genere nelle versioni stabili di HAProxy. Inoltre, HA-

Proxy è stato installato su dei sistemi che servono milioni di pagine al giorno e che hanno avuto un solo reboot in tre anni di attività.

Anche la sicurezza è un problema importante quando si effettua la messa in produzione di un load balancer. È possibile prendere contromisure a livello di sistema operativo, ma il load balancer rimane comunque esposto. Per questo motivo HAProxy è stato scritto con molta attenzione anche da questo punto di vista. La sua architettura limita significativamente l'impatto delle vulnerabilità, e offre spesso facili workaround. La sua imprevedibilità rende difficile lo sfruttamento dei bug, e se il processo va in crash il bug viene scoperto. Finora tutti i bug sono stati trovati analizzando i crash accidentali del processo. Inoltre, se avviato da root, il processo può chiudersi in una chroot per poi effettuare il drop dei privilegi.

I log forniscono molte informazioni e aiutano a mantenere un livello di sicurezza soddisfacente. I log sono normalmente inviati attraverso UDP perché dall'interno della chroot il socket di UNIX /dev/log è irraggiungibile e non deve essere possibile scrivere su un file. HAProxy fornisce anche la possibilità di controllare gli header tramite espressioni regolari. Parte delle richieste, così come gli header di richiesta e risposta, possono essere bloccati, permessi, rimossi, riscritti o aggiunti. Questa funzionalità è comunemente usata per bloccare richieste o encoding pericolosi e prevenire accidentali leak di informazioni dal server al client. Altre feature, come il controllo della cache, assicurano che informazioni sensibili non vengano salvate nella cache.

2.7.2 nginx

Nginx è un reverse proxy open source per i protocolli HTTP, HTTPS, SMTP, POP3 ed IMAP. Nginx può operare anche come load balancer, HTTP cache e web server. Gli obiettivi del progetto erano sviluppare un programma capace di operare con elevata parallelizzazione, alte performance, e un basso uso di memoria. Nginx utilizza un approccio asincrono e guidato da eventi per

gestire le richieste, a differenza del modello dell'HTTP server di Apache che utilizza un approccio orientato ai thread e ai processi. Questo approccio può fornire performance più prevedibili quando il carico è elevato.

2.8 Sistemi di virtualizzazione

La virtualizzazione è una tecnologia che ha preso piede negli ultimi anni e che permette di creare risorse virtuali sulle quali eseguire il software. In questa sezione si passano in rassegna i sistemi per la virtualizzazione delle risorse.

2.8.1 Xen

Xen è un hypervisor che utilizza un microkernel, fornendo servizi che permettono a più sistemi operativi di essere eseguiti sullo stesso hardware simultaneamente. Le prime versioni di Xen sono state sviluppate dal Computer Laboratory dell'Università di Cambridge. Attualmente, la comunità di Xen sviluppa e mantiene Xen come software libero e open source sotto la licenza GPL versione 2.

Xen è eseguito nello stato della CPU più privilegiato rispetto a ogni altro software in esecuzione sulla macchina. L'hypervisor si occupa di gestire la memoria e di schedulare le CPU di tutte le macchine virtuali (chiamate anche domini), oltre che lanciare il dominio più provilegiato (il cosiddetto dom0), che è l'unica macchina virtuale che di default ha accesso diretto all'hardware. L'hypervisor può essere gestito dal dom0, e da esso possono essere eseguiti domini non privilegiati (domU).

Il dom0 è tipicamente una versione di Linux o BSD. I domini utente possono essere sistemi operativi tradizionali (se il processore supporta la virtualizzazione hardware) oppure sistemi operativi para-virtualizzati, in cui il sistema operativo sa di essere in esecuzione all'interno di una virtual machine e non utilizza istruzioni privilegiate. Xen viene avviato tramite un bootloader, e subito dopo esso carica un sistema operativo paravirtualizzato nel dominio host.

2.8.2 KVM

KVM è un'infrastruttura di virtualizzazione per il kernel Linux che trasforma il kernel in un hypervisor, il cui supporto è stato integrato nel kernel ufficiale nel febbraio 2007, con la versione 2.6.20 del kernel [38]. KVM richiede un processore con capacità di virtualizzazione hardware. KVM è anche stato portato su FreeBSD [39] e Illumos [40] sotto forma di moduli del kernel. Inizialmente KVM supportava solo i processori x86, ma è stato in seguito portato anche alle architetture S/390, PowerPC, IA-64 e ARM.

Il supporto alla paravirtualizzazione è disponibile per i guest che possono utilizzare la API VirtIO con determinati dispositivi. È possibile paravirtualizzare una scheda Ethernet, un controller di I/O, un dispositivo per regolare l'utilizzo di memoria di un guest, e una interfaccia grafica VGA che utilizzi i driver SPICE o VMware.

KVM da solo non effettua alcuna emulazione. Esso semplicemente si occupa di esporre l'interfaccia /dev/kvm, con la quale un host userspace può impostare lo spazio di indirizzi della virtual machine, caricare un firmware che possa eseguire il sistema guest, mandare al guest I/O simulato, e rimappare il display del guest all'indietro verso l'host. L'host di riferimento con supporto a KVM consiste nel software QEMU, dalla versione 0.10.1 e successive. QEMU utilizza KVM quando disponibile, altrimenti effettua un fallback su una completa emulazione software.

2.8.3 VMware Workstation

VMware Workstation è un hypervisor che può essere eseguito sulle macchine x86-64 [41]. Permette agli utenti di creare una o più macchine virtuali su una singola macchina fisica e poi di usarle simultaneamente. Ogni virtual machine può eseguire un suo sistema operativo. VMware Workstation supporta il bridging di adattatori di rete preesistenti dell'host, e supporta anche la condivisione dei dischi e dei dispositivi USB con le macchine virtuali. Può

anche simulare dischi. Può montare immagini ISO esistenti in un disco ottico virtuale così che le macchine virtuali le vedano come dischi veri. Supporta anche hard disk virtuali, tramite file .vmdk.

VMware Workstation può salvare lo stato di una macchina virtuale (fare uno snapshot) in qualunque momento. Questi snapshot possono in seguito essere ripristinati per riportare la macchina virtuale allo stato precedente. VMware Workstation include la possibilità di designare più macchine virtuali come un insieme, permettendone la gestione come un oggetto singolo.

2.8.4 VirtualBox

VirtualBox è un Hypervisor per macchine x86, sviluppato inizialmente da Innotek GmbH, poi acquisita da Sun Microsystems nel 2008, acquisita a sua volta da Oracle nel 2010, la quale ne ha continuato lo sviluppo. VirtualBox può essere installato su un sistema host preesistente e poi creare e gestire macchine virtuali guest, ognuna con un proprio sistema operativo e ambiente virtuale.

I sistemi operativi host supportati includono Linux, OS X, Windows XP e successivi, Solaris e OpenSolaris. Esistono anche port per FreeBSD [42] e Genode [43]. I sistemi operativi guest supportati includono diverse versioni di Windows, Linux, BSD, OS/2, Solaris, Haiku e altri ancora.

Per ottenere la migliore esperienza possibile esiste un pacchetto chiamato "Guest Additions" che va installato sul sistema operativo guest. Esso consiste in device driver e applicazioni di sistema che ottimizzano il sistema operativo guest per migliori performance e usabilità. Dalla versione 4.3, i guest Windows possono trarre vantaggio dal WDDM driver incluso nel pacchetto delle Guest Additions, permettendo di abilitare il supporto alle Direct3D e usufruire di Windows Aero.

2.8.5 Vagrant

Vagrant permette di creare e configurare ambienti di sviluppo virtuali [44]. Esso può essere visto come un wrapper attorno a software di virtualizzazione come VirtualBox, VMware, KVM, LXC, e Docker, e attorno a software di gestione delle configurazioni, come Ansible, Chef, Salt, e Puppet.

Dalla versione 1.1 Vagrant non è più legato a VirtualBox e funziona anche con altri software di virtualizzazione, e supporta ambienti server come Amazon EC2 [45]. Vagrant è scritto in Ruby ma è utilizzabile anche in progetti scritti in altri linguaggi di programmazione.

Dalla versione 1.6 Vagrant supporta nativamente i container Docker, che funzionano come sostituto rispetto ad un sistema operativo completamente virtualizzato, il che riduce l'overhead. Vagrant supporta anche i plugin, e in particolare ne esiste uno che aggiunge il supporto a libvirt [46].

2.8.6 Docker

Docker è una soluzione di virtualizzazione a livello di applicazione, un progetto open source che automatizza il deployment delle applicazioni all'interno di container software, fornendo uno strato aggiuntivo di astrazione e automazione. Docker utilizza le feature di isolamento delle risorse del kernel Linux, come i cgroups e i namespaces [47], per permettere a container indipendenti di essere eseguiti all'interno di una singola istanza di Linux, evitando l'overhead tipico delle macchine virtuali.

Il supporto del kernel Linux ai namespaces isola la vista dell'applicazione da quella del sistema operativo, includendo l'albero dei processi, la rete, gli user ID, e i filesystem montati, mentre i cgroup del kernel forniscono l'isolamento delle risorse, inclusa la CPU, la memoria, l'I/O, e la rete.

Dalla versione 0.9, Docker include la libreria "libcontainer" per poter utilizzare direttamente le facilities di virtualizzazione fornite dal kernel Linux [48],

in aggiunta alla possibilità di utilizzare interfacce di virtualizzazione astratte attraverso libvirt, LXC, e systemd-nspawn.

Capitolo 3

Tecnologie utilizzate

Delle tecnologie descritte nello stato dell'arte per la costruzione di sistemi distribuiti, quelle effettivamente utilizzate sono descritte nei paragrafi seguenti.

Nell'implementazione iniziale del sistema si faceva uso di un framework IDL (Apache Thrift) per la definizione formale dell'interfaccia e per l'RPC. Inoltre, si utilizzavano gli Enterprise Java Beans per implementare lo strato di accesso ai dati, in modo che anche questo fosse gestito dall'application server, che su richiesta ne gestiva l'istanziazione. Inoltre, si faceva uso di JNDI per la localizzazione del servizio. Tuttavia questo modello si è rivelato essere troppo poco flessibile, e generava una dipendenza troppo forte tra le varie componenti del sistema, che non potevano essere distribuite a piacimento. Per questo motivo, nell'implementazione finale si è passati ad utilizzare un modello basato su scambio messaggi, multiprocesso, e multithread.

I linguaggi di programmazione utilizzati sono Java e Python. Java è stato utilizzato per la scrittura delle interfacce e per uno dei due moduli di accesso ai dati, mentre Python è stato utilizzato per l'implementazione del secondo modulo di accesso ai dati. L'utilizzo dei due linguaggi dimostra l'indipendenza dal linguaggio di programmazione per l'implementazione dei vari moduli. La scelta del linguaggio Python è stata fatta in modo tale da poter implementare il secondo modulo di accesso ai dati partendo da un progetto open source preesistente scritto in tale linguaggio. La scelta del linguaggio Java è invece

stata dettata dalla necessità di utilizzare un linguaggio ampiamente diffuso in ambito astronomico, ed integrabile con il resto dell'infrastruttura presente in IA2.

Per quanto riguarda il sistema di building, per il momento si fa uso del sistema di building interno dell'Integrated Development Environment (IDE) Eclipse, ma in futuro si utilizzerà Ant, in quanto Eclipse permette la generazione automatica del descrittore build.xml necessario al funzionamento di Ant.

Il formato in output fornito dai servizi VO consiste nel formato VOTable. Nel modulo di accesso ai dati scritto in Java, la libreria utilizzata per la serializzazione dei risultati nel formato VOTable si chiama STIL (Starlink Tables Infrastructure Library) [49]. In particolare, l'utilizzo di questa libreria ha permesso di generare una VOTable a partire da una sorgente SQL-based.

Il modulo di accesso ai dati scritto in Python invece si occupa personalmente della serializzazione nel formato VOTable. Il modulo Python è un fork del progetto open source "Simple-Cone-Search-Creator" disponibile su GitHub sotto la licenza BSD a 3 clausole [50]. La modifica effettuata al codice originario consiste nell'aggiunta al supporto per la messaggistica AMQP per l'ottenimento dei parametri della richiesta e per l'invio della risposta.

Il reverse proxy utilizzato nel progetto di tesi consiste nel software HAProxy, già descritto in precedenza nel capitolo sullo stato dell'arte. La scelta è ricaduta sul software HAProxy perché ricopre esattamente il ruolo di reverse proxy, mentre l'alternativa, il software nginx, oltre a poter lavorare come reverse proxy implementa anche altre funzioni, non necessarie nell'ambito della tesi.

Il protocollo di messaggistica utilizzato come middleware per lo strato di publish-subscribe è l'AMQP, anch'esso già descritto nel capitolo sullo stato dell'arte. In particolare, è stata scelta l'implementazione del protocollo fornita dal software RabbitMQ, scritto in Erlang ma con binding per diversi linguaggi

di programmazione, compresi i due linguaggi utilizzati, Python e Java [51]. In questo caso la scelta è ricaduta sul protocollo AMQP poiché le alternative non possedevano un'implementazione e una documentazione soddisfacente.

Per quanto riguarda il logging degli accessi, si pianifica di utilizzare la libreria tinylog, poiché non è necessario utilizzare implementazioni complesse e si ritiene più adatta una soluzione semplice e leggera.

Infine, il sistema di virtualizzazione utilizzato per l'esecuzione del software consiste nell'implementazione fornita da Xen, anche se una soluzione basata su KVM o Docker potrebbe rivelarsi più semplice da configurare e distribuire, e KVM è stato effettivamente utilizzato per la fase di testing. Tuttavia, nell'ambiente in cui il software verrà effettivamente utilizzato, il centro IA2, l'infrastruttura del sistema è basata su Xen.

Capitolo 4

Architettura del sistema

Il sistema distribuito realizzato è completamente modulare e scalabile, e può essere riassunto con la figura 4.1. La figura mostra come l'utente, effettuando la richiesta, incontra un reverse proxy che reindirizza la richiesta ad una delle interfacce, in modo da bilanciare il carico su di esse. Le interfacce si occupano poi di effettuare una richiesta RPC tramite scambio messaggi usando il protocollo AMQP, e consegnano il messaggio al broker, che è clusterizzato su un insieme di più nodi. A questo punto il broker consegna il messaggio all'apposito Data Access in ascolto, a seconda del servizio chiamato dall'utente, e individuato in base allo URL della chiamata. In aggiunta a questi moduli sono state sviluppate due ulteriori componenti: una componente per il logging, ovvero un processo che si occupa di rimanere in ascolto in attesa di nuovi messaggi di log, ed una componente che si occupa di fornire ai moduli di accesso ai dati la configurazione del servizio corrispondente. Il tutto è stato predisposto su tre nodi composti da macchine virtuali Xen.

4.1 Il Reverse Proxy

Tutte le chiamate dell'utente arrivano ad un reverse proxy che si occupa di smistare le chiamate al nodo appropriato. Nel caso specifico della tesi, il reverse proxy è un componente software la cui implementazione è fornita dal

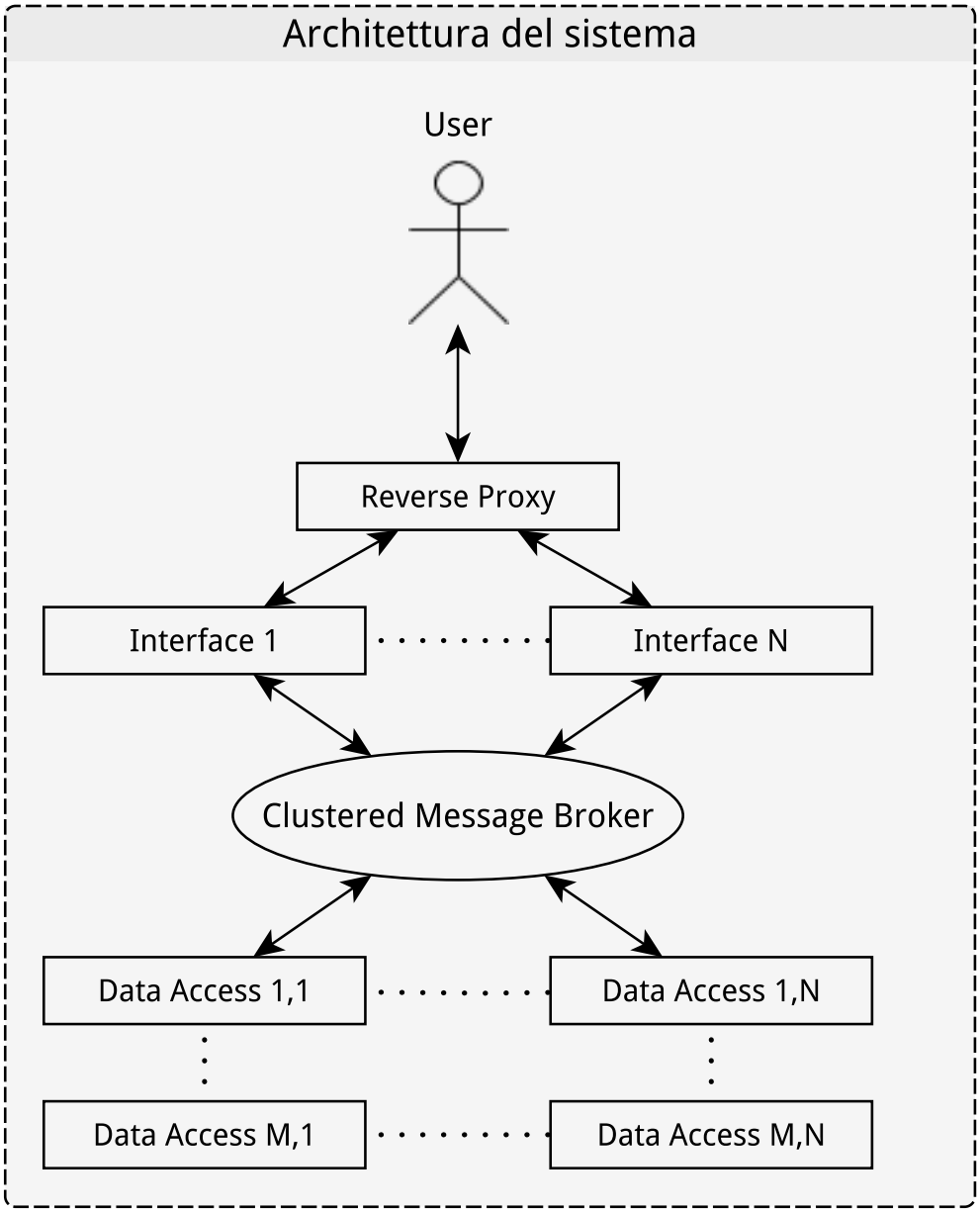

Figura 4.1: L'architettura del sistema

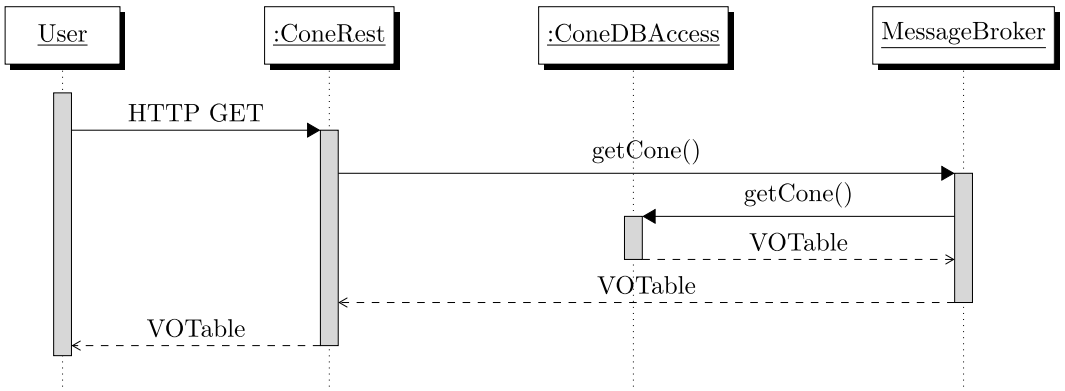

Figura 4.2: Il diagramma di sequenza in versione semplificata

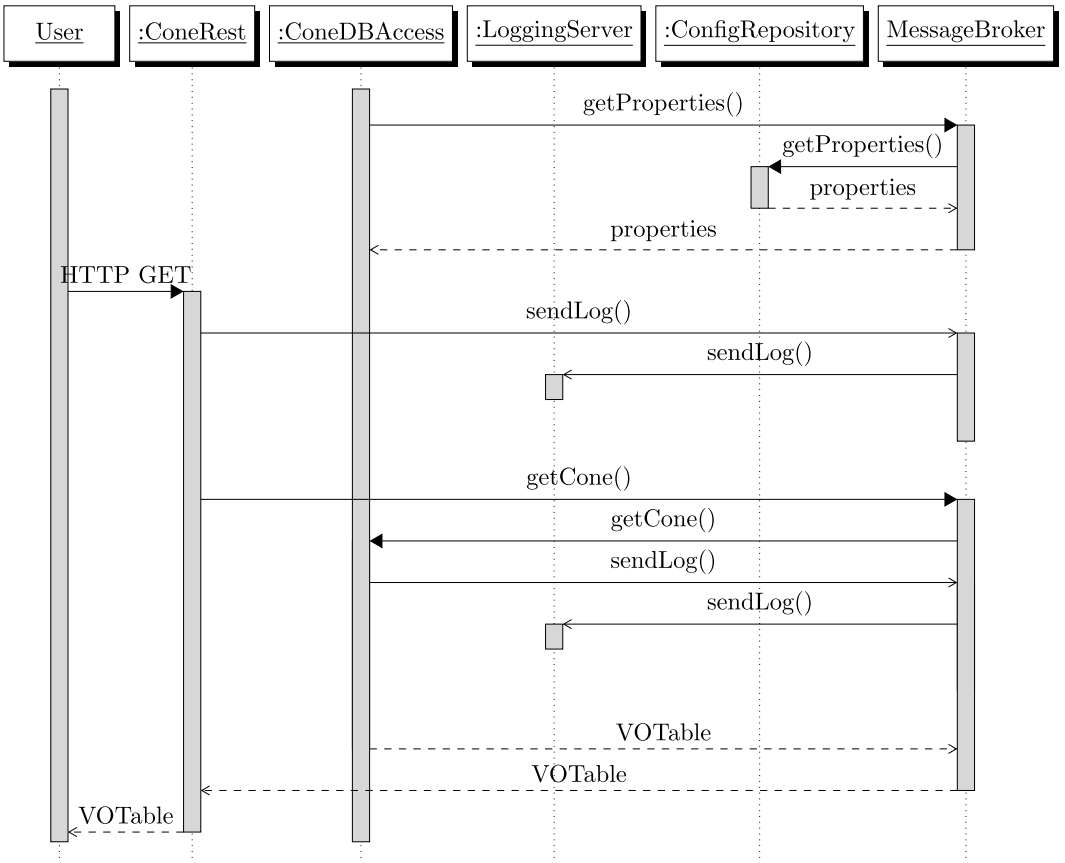

Figura 4.3: Il diagramma di sequenza

programma HAProxy. Un semplice file di configurazione permette di specificare i server verso i quali dovranno essere rispedite le richieste e quale algoritmo di bilanciamento del carico il software dovrà utilizzare.

4.2 L'Interfaccia

L'intefaccia verso l'utente del sistema consiste in un'applicazione web Java che si trova in esecuzione su un application server. Questa applicazione web non fa altro che restare in ascolto su un dato URL di base, al quale l'applicazione lato utente si occuperà di aggiungere l'ID del servizio desiderato ed eventuali altri parametri HTTP GET. L'interfaccia è generica e funziona così com'è per ogni tipologia di servizio da fornire. Più istanze della stessa interfaccia possono essere lanciate su host diversi in modo da utilizzare il reverse proxy per bilanciare il carico. L'interfaccia si occupa di processare la richiesta dell'utente generando un messaggio AMQP per il componente appropriato di accesso ai dati. Quest'ultimo viene individuato grazie all'ID del servizio presente nello URL della chiamata. Infine l'interfaccia rimane in attesa di una risposta, che viene rispedita all'utente appena disponibile.

4.3 Il Message Broker

Il message broker fornisce un middleware orientato ai messaggi per collegare l'interfaccia all'access data layer in modo che i due layer possano comunicare fra loro. La comunicazione avviene attraverso delle code di messaggi, che vengono create a partire dallo URL della chiamata. Eventuali servizi in ascolto che aspettano messaggi sulla coda appropriata possono recuperare il messaggio ed occuparsi di rispondere all'interfaccia sulla coda appositamente creata per la risposta. Questo meccanismo è equivalente ad una chiamata RPC, con la differenza che la chiamata avviene tramite scambio messaggi.

4.4 L'Access Data Layer

Lo strato di accesso ai dati è la componente che si occupa di implementare effettivamente il servizio: estrae i dati da una sorgente e li processa in modo da ritornare il risultato ricercato dall'utente. La sorgente dei dati non è prestabilita, così come non lo è il linguaggio di programmazione utilizzato per implementare un nuovo servizio. Grazie al paradigma di scambio messaggi, il message broker si interpone tra i due strati dell'architettura, risultando in una maggiore modularità e indipendenza fra le varie componenti. Nell'ambito specifico della tesi, sono stati scritti due diversi componenti di accesso ai dati, per dimostrare la modularità sopra descritta. Uno dei due componenti è scritto in Java e accede ad un database MySQL tramite JDBC, mentre l'altro è scritto in Python e accede ad un file CSV. Entrambi possono essere in esecuzione contemporanea per fornire risposte per lo stesso servizio o per servizi diversi. Nel caso in cui entrambi rispondano allo stesso servizio, è opportuno verificare che le diverse sorgenti dati forniscano gli stessi risultati.

4.5 Il Logging Server

Il Logging Server consiste in un listener AMQP che rimane in ascolto su una coda di messaggi creata specificamente per gestire i messaggi di log. Si tratta a tutti gli effetti di un wrapper per la libreria tinylog, a cui è stato aggiunto il supporto per ricevere messaggi di log attraverso il protocollo AMQP. Al Logging Server è stato affacciato un Logging Client che fornisce dei metodi di convenienza per inviare i messaggi di log tramite AMQP.

4.6 Il Config Repository

Il Config Repository è un altro listener AMQP che si occupa di restare in ascolto su una coda di messaggi specifica e fissata nel tempo. Il Config Repository è stato pensato per contenere le configurazioni di tutti i servizi

supportati, e per fornire queste configurazioni a runtime ai moduli di accesso ai dati che ne fanno richiesta quando questi vengono avviati. L'unico parametro necessario per lo scambio della configurazione consiste nell'ID del servizio. Come risposta viene creato un oggetto che rispecchia un file di properties e serializzato tramite JSON, di cui un esempio è mostrato di seguito.

```
{
  "dbName": "wings",
  "tableName": "gasphotb",
  "raColumn": "ra",
  "decColumn": "decl",
  "dbUnit": "Degrees",
  "dbUser": "root",
  "dbPassword": "*****",
  "dbHost": "localhost",
  "dbPort": "3306"
}
```

Capitolo 5

Test e valutazione del sistema

Il testing e la valutazione del nuovo sistema rispetto a quello precedente è stato effettuato tramite il tool 'siege', un HTTP/HTTPS stress tester [52]. Innanzitutto è stato necessario verificare che la risposta alla query fosse identica da parte di entrambi i sistemi. Una volta accertato il corretto funzionamento da parte del nuovo sistema si è provveduto a lanciare il benchmark. Di seguito i risultati, rispettivamente del vecchio e del nuovo sistema.

```
$ siege <endpoint> -t 1 -b
```

	VO-Dance	VOBall	(Unit)
Transactions:	453	1257	hits
Availability:	100.00	100.00	%
Elapsed time:	59.63	59.26	secs
Data transferred:	0.35	4.80	MB
Response time:	1.95	0.70	secs
Transaction rate:	7.60	21.21	trans/sec
Throughput:	0.01	0.08	MB/sec
Concurrency:	14.80	14.90	
Successful transactions:	453	1257	
Failed transactions:	0	0	
Longest transaction:	3.08	1.68	
Shortest transaction:	0.38	0.15	

Come si può notare dai risultati, in particolare osservando il numero di transazioni effettuate nello stesso arco di tempo, il nuovo sistema è quasi tre

volte più veloce rispetto al precedente. Questo risultato è coerente con le aspettative, in quanto il nuovo sistema è stato distribuito su tre nodi diversi, mentre il vecchio sistema era vincolato ad un solo nodo, in quanto esso non supporta la distribuzione del calcolo su più nodi.

Capitolo 6

Conclusioni e sviluppi futuri

In questa tesi è stato proposto e sviluppato un sistema distribuito per la pubblicazione di servizi VO. In particolare sono stati implementati il modulo di interfaccia e due diversi moduli per l'accesso ai dati, che implementano due servizi VO di tipo Simple Cone Search. Ad essi sono stati aggiunti un reverse proxy ed un message broker, per garantire alta disponibilità e scalabilità sia orizzontale che verticale. Infine sono stati sviluppati un modulo per il logging e un modulo per la gestione della configurazione dei servizi. Questo tipo di architettura ha permesso di superare le problematiche riscontrate nel precedente sistema di pubblicazione, ed in particolare il problema delle performance, come dimostrato dallo stress test effettuato su entrambi i sistemi. Inoltre, il nuovo sistema di pubblicazione fornisce una base per l'implementazione di nuovi protocolli, un possibile sviluppo futuro. Si penserà infine all'implementazione di moduli aggiuntivi, ausiliari alla pubblicazione dei servizi, come un generatore di VOTable e un'interfaccia di amministrazione del sistema.

Bibliografia

[1] C. Arviset, S. Gaudet, and IVOA TCG. The IVOA Architecture. In P. Ballester, D. Egret, and N. P. F. Lorente, editors, *Astronomical Data Analysis Software and Systems XXI*, volume 461 of *Astronomical Society of the Pacific Conference Series*, page 259, September 2012.

[2] R. Plante, R. Williams, R. Hanisch, and A. Szalay. Simple Cone Search Version 1.03. IVOA Recommendation 22 February 2008, February 2008.

[3] F. Ochsenbein, M. Taylor, R. Williams, C. Davenhall, M. Demleitner, D. Durand, P. Fernique, D. Giaretta, R. Hanisch, T. McGlynn, A. Szalay, and A. Wicenec. VOTable Format Definition Version 1.3. IVOA Recommendation 20 September 2013, September 2013.

[4] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. `http://www.w3.org/TR/wsdl20/`, 2007.

[5] Apache Thrift Libraries. `https://thrift.apache.org/lib/`.

[6] Apache Avro Supported Languages. `https://cwiki.apache.org/confluence/display/AVRO/Supported+Languages`, 2015.

[7] Apache Avro IDL. `http://avro.apache.org/docs/current/idl.html`.

[8] Welcome to Apache Etch. `https://etch.apache.org/`.

[9] Third-Party Add-ons for Protocol Buffers. `https://github.com/google/protobuf/wiki/Third-Party-Add-ons`.

[10] Franca goes Eclipse. http://eclipse.org/proposals/modeling.franca/, 2013.

[11] ZeroC Ice Licensing Information. https://zeroc.com/licensing.html.

[12] ZeroC Ice Language Support. https://zeroc.com/languages.html.

[13] Differences between Ice and CORBA. http://web.archive.org/web/20130318043237/http://www.zeroc.com/iceVsCorba.html.

[14] JSR 318: Enterprise JavaBeans 3.1 Specification. https://jcp.org/en/jsr/detail?id=318.

[15] JSR 342: Java Platform, Enterprise Edition 7 (Java EE 7) Specification. https://www.jcp.org/en/jsr/detail?id=342.

[16] The Scala Actors Migration Guide. http://docs.scala-lang.org/overviews/core/actors-migration-guide.html.

[17] Sun releases Jini with open-source license. http://www.cnet.com/news/sun-releases-jini-with-open-source-license/.

[18] Fabrizio Montesi, Claudio Guidi, and Gianluigi Zavattaro. Service-Oriented Programming with Jolie. In Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, editors, *Web Services Foundations*, pages 81–107. Springer New York, 2014.

[19] Apache Ant Frequently Asked Questions. https://ant.apache.org/faq.html.

[20] .NET Maven Plugin. http://doodleproject.sourceforge.net/mavenite/dotnet-maven-plugin/index.html.

[21] Maven Native C/C++ plugin. http://maven-nar.github.io/index.html.

[22] Apache Maven plugins. https://maven.apache.org/plugins/index.html.

[23] MojoHaus Maven plugins. http://www.mojohaus.org/plugins.html.

[24] Introduction to the Build Lifecycle. https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html.

[25] Maven 2 Central Repository. http://repo1.maven.org/maven2/.

[26] OASIS UDDI Specifications TC - Committee Specifications. https://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm.

[27] XMPP History. http://xmpp.org/about-xmpp/history/.

[28] XMPP Servers. https://xmpp.org/xmpp-software/servers/.

[29] XMPP Clients. https://xmpp.org/xmpp-software/clients/.

[30] XMPP Libraries. https://xmpp.org/xmpp-software/libraries/.

[31] MQTT V3.1 Protocol Specification. http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.

[32] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf.

[33] Java(TM) Message Service Specification Final Release 1.1. http://download.oracle.com/otndocs/jcp/7195-jms-1.1-fr-spec-oth-JSpec/.

[34] Reasons to prefer logback over log4j. http://logback.qos.ch/reasonsToSwitch.html.

[35] Apache Log4j 2. http://logging.apache.org/log4j/2.x/.

[36] Tinylog Benchmarks. http://www.tinylog.org/benchmark.

[37] HAProxy - They use it. http://www.haproxy.org/they-use-it.html.

[38] Virtualization support through KVM. http://kernelnewbies.org/Linux_2_6_20#head-bca4fe7ffe454321118a470387c2be543ee51754.

[39] Porting Linux KVM to FreeBSD. http://www.freebsd.org/news/status/report-2007-07-2007-10.html#Porting-Linux-KVM-to-FreeBSD.

[40] KVM on illumos. http://dtrace.org/blogs/bmc/2011/08/15/kvm-on-illumos/.

[41] Processor Requirements for Host Systems. http://pubs.vmware.com/workstation-9/index.jsp?topic=%2Fcom.vmware.ws.get_started.doc%2FGUID-BBD199AA-C346-4334-9F56-5A42F7328594.html.

[42] VirtualBox on FreeBSD. https://wiki.freebsd.org/VirtualBox.

[43] Release notes for the Genode OS Framework 14.02. http://genode.org/documentation/release-notes/14.02.

[44] Introducing Vagrant. http://www.linuxjournal.com/content/introducing-vagrant.

[45] Mitchell Hashimoto. Vagrant: Up and Running. In *Vagrant: Up and Running*, page 13. O'Reilly Media, 2013.

[46] Vagrant Libvirt Provider. https://github.com/pradels/vagrant-libvirt.

[47] Docker Kernel Requirements. http://web.archive.org/web/20140821065734/http://docker.readthedocs.org/en/v0.7.3/installation/kernel/.

[48] Docker 0.9: Introducing execution drivers and libcontainer. http://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/.

[49] M. B. Taylor. TOPCAT & STIL: Starlink Table/VOTable Processing Software. In P. Shopbell, M. Britton, and R. Ebert, editors, *Astronomical Data Analysis Software and Systems XIV*, volume 347 of *Astronomical Society of the Pacific Conference Series*, page 29, December 2005.

[50] Simple Cone Search Creator. https://github.com/tboch/Simple-Cone-Search-Creator/.

[51] Getting started with RabbitMQ. https://www.rabbitmq.com/getstarted.html.

[52] siege - An HTTP/HTTPS stress tester. http://linux.die.net/man/1/siege.