DIRAC LIGHTWEIGHT INFORMATION AND MONITORING SERVICES USING XML-RPC AND INSTANT MESSAGING

I. Stokes-Rees*, University of Oxford, Oxford, UK
A. Tsaregorodtsev, V. Garonne,
Centre de Physique des Particules de Marseille, Marseille, France

Abstract

This paper presents recent work on a scalable, lightweight approach for distributed information and monitoring systems done for the LHCb experiment's DIRAC grid software package. Two complementary systems are presented, one based on a layered, DNS-like information service, and the other a monitoring mechanism using instant messaging for *ad hoc* networks which are formed in a grid environment. The paper outlines the requirements, design architecture and features of the implementation. The performance results of these two systems in the recent LHCb 2004 Data Challenge are presented.

INTRODUCTION

In a grid environment it is necessary to manage the discovery, monitoring, coordination, and configuration of a distributed set of independent services. The DIRAC grid architecture [1, 8], developed for the CERN LHCb [4] experiment, does this via an instant messaging framework and a lightweight interface for service information discovery, together called the Information and Monitoring Services.

This paper will discuss the features of these services and how they have performed during the 2004 LHCb Data Challenge (DC04). Discussion of the overall DIRAC architecture will not be covered, as it is discussed elsewhere [9]. The first part of the paper covers the Information Service API, providing a motivation for its development and a brief comparison against alternative existing information services. The second part will discuss opportunities for using instant messaging technology in a grid environment for component monitoring and control. Finally the paper will mention future development plans for these services.

DIRAC INFORMATION SERVICE

When working with large numbers of dynamic collaborating components, possibly with replication either on the same host or across a set of hosts, coordinating the configuration and information access for each of these components and between components is a difficult task. It was identified that every Service, Agent, and Client within DIRAC required a uniform API which would provide:

- Local configuration and information
- Global (system wide) configuration and information
- Remote component configuration and information
- Configuration and information sharing
- Overriding of global settings
- Ease of deployment
- Ease of updates
- Robustness
- Simplicity

This is closely related to the classic Name Service problem, addressed in other contexts by DNS (Domain Name Service) [6], LDAP (Lightweight Directory Access Protocol) [11], and UDDI (Universal Description, Discovery, and Integration) [10], to name a few. It was not without considering these existing approaches that the DIRAC Information Service was developed. Of these alternate existing systems, DNS came the closest to providing a simple, de-centralised system that did not require the installation and configuration of a central information server, or have particular pre-conceptions concerning the information contained by the service or the method of service access. Unfortunately DNS still presented a sufficient level of complexity from both the implementation and end user perspective to warrant the development of a new service. However, many design ideas are borrowed from the DNS architecture, such as iterative navigation, hierarchical information, replication, and caching.

In an effort to utilise pre-existing software wherever possible, and given the extensive usage of the Python scripting language within LHCb, the configuration file format supported by the standard Python module *ConfigParser* [5] was taken as a starting point. The associated file format provides categorized name/value pairs. Figure 1 illustrates a simple example INI file (the name is taken from the Microsoft Windows INItialisation files which popularised this format). The categories are called "sections", which consist of zero or more "options". Each option has a text value.

The simplicity of this format means non-expert users can easily modify configuration files. As well, it presents an information data model which is conceptually easy to grasp. The goal was to present information to a software component as if it had come from a single local INI file. The basic interface is borrowed directly from the ConfigParser module, as listed below (the optional `[src]` parameter will be discussed later):

To clarify the distinction between the local object which exposes this information source API and a remotely accessible service, two distinct classes were created: `LocalInformationService` and `InformationService`, respectively. The intention

* also Marie Curie Fellow at CPPM, Marseille, France

```
[ServiceA]
ServiceName = DIRAC Job Matcher
CEUniqueIds = in2p3.fr/pbs-short

[InfoService]
List = /etc/site-config.ini     \
       http://lbnts2.cern.ch    \
       http://marsanne.in2p3.fr
```

Figure 1: *Example INI file*

```
void    set     (section, option, value, [src])
value = get     (section, option, [src])
list  = options (section, [src])
list  = sections([src])
```

Figure 2: *Information Service base API*

is that on a semantic level the APIs to these two objects will be identical.

API Features

A LocalInformationService object can be passed a number of information sources when it is created. This ordered list represents the hierarchy of sources which will be queried in order, either until a requested item is found or an exception returned (indicating the item does not exist) once all sources have been attempted.

This list of sources can be composed of a mixture of local files and remote sources. Local files are read directly into memory and not referenced again, while remote sources are queried only when necessary. This approach implies file based information is static and a snapshot is taken at object creation time, while remote information can be dynamic and subsequent requests may return different results. A mechanism exists to copy results from remote sources, placing a snapshot of those requested items in memory in the local object, avoiding subsequent calls to the remote service, but sacrificing the ability to catch changes to remote information.

There is also the option to create a LocalInformationService without any information sources, and simply add information to the object during program execution, or add a list of sources at a later point. Similarly it is possible to change the list of remote sources, meaning a single LocalInformationService object can act as an interface to request information dynamically from any remote source. This functionality explains the use of the optional `[src]` parameter in the API, which makes the object a stateless adapter for interfacing to a remote information source exposing the Information Service API.

The Information Service has been designed to be sufficiently simple that any Service or DIRAC component can expose its local configuration information via an XML-RPC interface and therefore be accessed as a remote Information Service.

Implementation, Operational Experience, and Performance

An InformationService object contains a LocalInformationService object, or equivalent representation, and exposes the contents of that object via a remotely accessible API. In the case of DIRAC, this object provides the stated API via an Internet accessible multi-threaded XML-RPC interface, using HTTP. The actual information is contained in a database for persistency, making use of 3-tuples of (section,option,value), where (section,option) form a joint key (i.e. must be unique within the table). This service is supported by a multi-threaded database connection pool which allows a pre-defined number of simultaneous queries to be supported.

During DC04, it was found that a configuration with a single central information service and a set of local INI files for each component allowed the efficient distribution and management of system information. In this way, the central Information Service provided system-wide "global" information, such as the location of other services, grid-enabled storage nodes, or configuration parameters. At each site a site-wide configuration file was used, coupled with a component specific configuration file. These three sources (component, site, global) allowed Clients, Agents, and Services to self-configure and inter-operate, and provided the flexibility to override information if necessary (*e.g.* by utilising a local value in preference to a global value).

By combining replication of Information Services, time-outs, retries, fail-overs, caching, and block queries, a robust grid information infrastructure was established. It supported thousands of jobs executing simultaneously. During DC04 the Information Service was constantly being queried, with peaks of 300 requests per minute, however the relational database server began to saturate above rates of 40 queries per second.

It should be noted that while the initial design was meant to support dynamic information updates and "live" information regarding the state of services, in fact it was found that the infrastructure was primarily used as a source of configuration information, utilised primarily when a component was created, or when a particular operation (such as a file transfer) was performed. The vision of every component providing real-time dynamic state information via this infrastructure has not been heavily exercised, although in principle is possible. The reason for this is related to the development of the complementary instant messaging infrastructure which appears to be better suited to this task, and which is discussed in the following section.

INSTANT MESSAGING

A grid computing environment is extremely dynamic and unpredictable. Data is created, replicated, modified, and deleted continuously, and a "grid job" may be executed on one or more processors at a random grid computing site at some undefined point in time after it is submitted. The emerging concept of "Grid Services" [2] also suggests an explosion in the number of dynamic services, constantly being created, accessed, and destroyed by jobs. Given the necessity of firewalls and the prevalence of Network Address Translation (NAT) to partition off a computing cluster or node onto a sub-net it can be extremely difficult to gain access to active processes or "live data" (data which is in the process of being created) for real-time information concerning their state or for control. While the grid middleware may provide some generic process discovery and control mechanisms, these typically are too coarse grained to be especially useful. Furthermore, a grid computing environment experiences much higher levels of service inaccessibility than what is found in homogeneous, centrally managed, single site computing facilities.

Instant messaging provides a mechanism to connect grid components and users in a peer-to-peer fashion, transcending firewall and NAT issues, via a portable instant messaging address. Certain instant messaging infrastructures also provide message buffering, thereby protecting communications from network outages, overloads, and restarts. These features drew the LHCb grid software group to investigate the potential of using instant messaging to resolve the aforementioned problems.

The Jabber [3]/XMPP [7] instant messaging protocol was selected due to its simplicity (XML based with only three message types), maturity and library availability for all languages and platforms (including numerous clients and servers). The sole drawback was the use of central servers which act as messaging hubs, however this provided the advantage of "thin-clients".

Instant Messaging within DIRAC

The original application of instant messaging within DIRAC was to provide asynchronous, buffered messaging between Services. Each Service Class was assigned a single authentication credential (User account), and each Service instance utilised a unique Resource name to provide a single unique address for that instance. This mechanism decoupled Services from each other and allowed them to be stopped, restarted, and even moved to different hosts during live operation ("service hot-swapping"). This was critical for robustness of the overall system and maintenance of individual services. Given the number of Services was small (5-20), and the communication between the Services limited, there were no problems with bottlenecks at the central XMPP server seen with this approach.

The next step was to introduce instant messaging for state monitoring of Agents. By using the "chat room" functionality of instant messaging, an *ad hoc* messaging hub could be created. Agents could connect to an "Agent Chat Room" and publish progress information as chat room messages, and the room Roster list acted as an inventory of online Agents. By using custom status fields the Roster also provided information regarding where the Agent was running (host name, directory, process number). This was critical when mis-behaving Agents were discovered. Again, given the number of Agents was initially low (10-100), this operated well and allowed a system administrator to use a standard GUI Client to connect to the same Chat Room and monitor Agent status.

This naturally led to the question of introducing instant messaging to each Job. The intention was to provide job-level live monitoring. Due to the fact that the number of active jobs was two orders of magnitude greater than the number of Agents (1000-10,000 active jobs), it was discovered that thousands of automated Instant Messaging clients connecting to a single XMPP server or chat room resulted in a Distributed Denial of Service (DDoS) attack. The messaging load saturated the network connection, caused the server to consume all available memory, and overloaded the server processor. The XMPP server software was running on the same server as the other DIRAC software services and therefore paralysed the entire system. On the Client side, the XMPP connection was not done in a separate process or thread, so the blocking resulted in stalled processes.

This experience indicated that the XMPP server had to be independent or sand-boxed so as to not overwhelm other services on the same host, and that any DIRAC components making use of XMPP client-side connections had to do so in a non-blocking manner – that is, either in a separate thread or fork, and with appropriate timeouts. A memory leak in the JabberD2 server software meant that above 1000 simultaneous connections the process would grow exponentially in size and eventually crash. This has since been fixed, but at the time the integration of DIRAC with the LHC Computing Grid made it necessary to operate one Agent per Job, meaning that even limiting Instant Messaging usage to Services and Agents would result in excessive connections.

Within the development branch of DIRAC, another application of interest has been implemented. This is to make use of the `<iq>` messages to provide RPC functionality through to Agents and Jobs, allowing them to be remotely controlled, and to provide access to data local to a Job. The initial implementation provides just basic process control and small file transfer, however in principle a much richer level of RPC interactivity is possible. Utilisation of the standard `<message>` messages would also allow interaction with Services, Agents, and Jobs using standard Jabber/XMPP GUI clients, however this has not yet been investigated in depth.

Operational Experience

For DC04 it was only possible to utilise instant messaging for inter-service communication. The way instant

messaging was used with Agents and Jobs did not scale to the loads DC04 placed on the server, and which the server software could support. This base inter-service functionality was valuable, but it was clear that Jabber/XMPP instant messaging could introduced a bottle neck and single point of failure. The DDoS problem was not unique to Jabber, but still must be considered. Instant messaging communications must be kept to manageable levels. It is hoped that more recent versions of the JabberD server and changes to how Instant Messaging is utilised within DIRAC will allow Agents and Jobs to utilise XMPP for monitoring and interactive control.

An important issue is the authentication and authorization of XMPP Entities. The current system can make use of SSL/TLS to encrypt session contents, however no open-source servers currently support client authentication using digital certificates. All authentication is username/password based, and relies on a secure server, rather than end-to-end message security. For automated XMPP Entities the current servers either require hardcoded passwords, or the removal of password based security.

FUTURE DIRECTIONS

Information Service

The Information Service architecture has proved to be functional, flexible, and scalable. The DIRAC team intends to implement automated replication with both pull and push modes for the Information Service to improve robustness. A security infrastructure which would associate access and operation based rights with information entries or sections would facilitate publishing and updating entries. Information lifetimes would also be beneficial both for expiring outdated entries, and for cache management. Alternatives to the 3-tuple relational database persistent Information Service are planned to compare performance and features.

Instant Messaging

There are a great number of potential applications of instant messaging technology within grid computing. LHCb plans to improve the scalability of the instant messaging framework and investigate ways to avoid the DDoS problem. It is essential that x509 digital certificate based authentication can be incorporated, and work is under way on this at Brookhaven National Laboratory which is being followed closely. Three key areas of interest exist: logging, job coordination, and interactivity. An XMPP Entity can be setup to act as a logging end point, and all messages sent to it are timestamped, marked with the message source, and appended to a persistent store. Chat rooms can be used, as mentioned earlier, as a messaging hub to form an *ad hoc* network to complete a parallel computation workflow within a "meta-job". Finally, the `<iq>` functionality provides great promise for job steering and job interactivity. Users could build their own interfaces into the jobs and have the jobs notify them via instant messages when they start, as they progress, and at key way points, giving the option to change parameters, suspend or cancel jobs, and access local job files.

CONCLUSIONS

The DIRAC Information and Monitoring Services have proved to be simple, lightweight, and functional. Their successful use during LHCb DC04 has resulted in a design and implementation which can be extended to provide more general functionality and greater robustness in the future. For both XML-RPC and Instant Messaging secure access channels must be established. Future work will involve improving the scalability of both systems and adding additional interaction mechanisms. Attention will also be given to performance benchmarking and comparison against alternatives. The subsequent versions of these services will be utilised for the LHCb 2005 Data Challenge.

REFERENCES

[1] DIRAC Grid Software. `http://dirac.cern.ch/`.

[2] I Foster, C Kesselman, J Nick, and S Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG*. Global Grid Forum, 22 June 2002.

[3] Jabber Software Foundation. `http://www.jabber.org/`.

[4] LHCb. `http://lhcb.web.cern.ch/lhcb/`.

[5] Python Standard Library. ConfigParser Module. `http://www.python.org/doc/current/lib/module-ConfigParser.html`.

[6] P. Mockapetris. "RFC 1034: Domain Names - Concepts and Facilities. `http://www.ietf.org/rfc/rfc1034.txt`, November 1987.

[7] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. `http://www.ietf.org/rfc/rfc3920.txt`, October 2004.

[8] Andrei Tsaregorodsev et al. DIRAC - Distributed Implementation with Remote Agent Control. In *Proceedings of Computing in High Energy and Nuclear Physics (CHEP)*, April 2003.

[9] Andrei Tsaregorodsev et al. DIRAC - The Distributed MC Production and Analysis for LHCb. In *Proceedings of Computing in High Energy and Nuclear Physics (CHEP)*, October 2004.

[10] Universal Description, Discovery and Integration. `http://www.uddi.org`.

[11] M. Wahl, T. Howes, and S. Kille. RFC 2251: Lightweight Directory Access Protocol (v3). `http://www.ietf.org/rfc/rfc2251.txt`, December 1997.