A Virtual Registry For Wide-Area Messaging

Mark Baker
ACET, University of Reading, UK
mark.baker@computer.org

Matthew Grove
DSG, University of Portsmouth, UK
matthew.grove@port.ac.uk

Abstract

Tycho is a reference implementation of a combined extensible wide-area messaging framework with a built in distributed registry for publishing and discovering remote endpoints. This paper describes the architecture of Tycho as well as the design and functionality of its Virtual Registry (VR). We explain why we have designed Tycho to reuse existing infrastructure where possible and the advantages of a messaging system with a built in registry. We discuss the pluggable design of the VR and how that can be used to provide inter-operation with other systems. After highlighting our innovative use of the Internet Relay Chat protocol within the VR, we present the results of a series of tests designed to measure the performance of the VR. We then compare the performance of Tycho with R-GMA and Globus's MDS4, which have similar functionality. The paper concludes with a number of observations about Tycho's VR performance, functionality, and how these can be enhanced in the future.

1 Introduction

Tycho is an extensible reference implementation of a wide-area asynchronous messaging system with a built in distributed registry. This combination allows Tycho to provide a range of key services for wide-area distributed applications that can essentially publish and discover endpoints, as well as exchange information without the need for the developer to use multiple libraries. Existing solutions typically use one of many communication mechanisms, for example SOAP [1] or GridFTP [2] coupled with a separate registry such as UDDI [3] or Globus's MDS4 [4] to provide service discovery. The objective of the Tycho project is to produce a combined asynchronous messaging API and registry system that frees developers from the need to assemble their application from a range of potentially diverse middleware offerings, which should simplify and speed application development and more importantly allow developers to concentrate on their domain of expertise. We believe that this combined approach, i.e. providing a messaging API and a virtualized registry that utilises existing infrastructure where possible is novel.

The design of Tycho is such that even though a default registry and protocols are provided, the system can be easily adapted to incorporate other registry technologies and transport protocols. Our design philosophy has been to keep the core of Tycho relatively small, simple and efficient, so that it has a minimal memory foot-print, it is easy to install, and is capable of providing robust and reliable services. More sophisticated services can then be built on Tycho's core and provided via libraries and tools to applications. We limit the use of external dependencies, such as libraries, in order to simplify the use of Tycho. Currently, we have incorporated all of its functionality within a single Java Jar with the only requirement being a Java 5 JDK for building and running Tycho applications.

Previous work on Tycho [5] has concentrated on measuring and optimising the message-passing performance of the system. We have run a series of tests on both the Tycho messaging sub-system and the NaradaBrokering [6] framework. Our results showed that Tycho and scales well, both with increasing message sizes and number of simultaneous clients without requiring the Java heap size to be increased.

In this paper we compare and contrast Tycho's virtual registry to two other popular systems with similar characteristic; namely R-GMA, part of the gLite middleware, and MDS4 which is part of the Globus Toolkit. In Section 2, we outline the architecture of R-GMA and MDS4. The architecture of Tycho's virtual registry is outlined in Section 3. In Section 4, we describe the performance tests we undertook on the three systems and then discuss the results obtained. Finally, in Section 5, we summarise and conclude the paper, present some details on how Tycho is currently being used and then outline some future work.

2 Related Work

Registries (or discovery services) form an integral part of all distributed systems. A registry is a unifying component that allows entities to publish their presence, which others can then later search for and bind with. In modern distributed systems, with large numbers of clients and resources, there is likely to be the need to publish many entities within the registry. Consequently a registry must have some intelligence, as it will not be appropriate for a registry to publish a list of all known entities held; clients must be able restrict the number of results by providing some attributes which the registry will use to filter the matching results. It is also clear that a registry needs a scalable architecture itself, as a single server will become a bottleneck if there are millions of entities held in its database with large numbers of clients regularly undertaking searches.

Two popular systems, which provide registries as part of their services, are gLite, which uses R-GMA and the Globus Toolkit, which uses MDS4. The aim of both of these systems is to simplify the process of binding grid services together by providing a standards-based approach publishing, searching and binding clients to their desired services. In this section we describe these two systems, outline their architecture and describe their features.

2.1 R-GMA

R-GMA [7] was originally developed within the European DataGrid Project [8] as a grid information and monitoring service. R-GMA uses a relational model to search, (using SQL as a query language) and describe the monitoring information it collects. It is based on a consumer/producer paradigm with client data being stored in a directory service, which presents the information as a virtual database. R-GMA uses a global schema, to define tables for storing data, which can be updated to contain project specific values. The current stable release of R-GMA (in gLite 1.5) is based on Java servlets; a more advanced version is under development using Web Services and SOAP for messaging.

The current stable release of R-GMA uses a set of Java servlets to implement the registry and provide services to producers and consumers. A producer's client publishes data (tuples) via the producer servlet, when a consumer's client performs a query; the consumer servlet performs the query and receives the tuples marked up in XML, it then stores responses in a queue. The consumer's client can then 'pop' the tuples from this queue to receive them. In this way the clients of the producer and consumer never interact directly, with the servlets mediating the process of publishing, querying and receiving data.

The servlets use a MySQL database to store tuples and meta information such as the schema. It is possible to specify that tuples are stored in memory rather than the persistent database when the producer publishes them. Republishers can manually be configured to allow remote queries from other R-GMA installations.

2.2 MDS4

The Globus Toolkit's [9] Monitoring and Discovery Service (MDS version 4) is a WSRF [10]-based implementation of an information and registry service. MDS4 provides a framework that can be used to collect, index and expose data about the state of grid resources and services.

MDS4 uses an index service for gathering resource information and provides a WSRF interface for clients to query and subscribe to information collected by the service. MDS4 is typically used for monitoring resources within a virtual organisation. The MDS4 functionality is provided by an aggregator framework, which is based on two services; an index service for collecting and discovering information about resources and a trigger service, which can be configured to perform actions based on resource information. MDS4 uses a fixed schema to ensure compatibility between its components, every participating MDS must be configured to use the same schema. The index information itself is stored in memory.

Grid services can publish resource properties into MDS4; these properties are defined in the service's WSDL. An MDS4 client can query the index service using an XPath query that will retrieve matching resource properties marked up as XML. MDS4 uses a soft state consistency model to renew information published resources at a configurable interval.

2.3 Summary

In this section, we have briefly outlined two systems, which contain registries for publishing information about resources. MDS4 and R-GMA can be used more generically for services other than monitoring. The fixed global (system wide) MDS4 schema limits the type of information that can be published into the MDS. R-GMA also uses a global schema, but it is designed to be simple to configure for new types of information. This facet could be described as a project wide schema as it is used by every client connected to the R-GMA virtual database. In Tycho we allow the schema to be defined on a per client basis at run time. This removes the need for each Tycho instance participating in a system to be manually configured to ensure the correct schema is being used, this is explained in more detail in Section 3.1

3 Tycho's Architecture

Tycho is a Java-based wide-area messaging and registry framework based on a publish, subscribe and bind paradigm. Tycho consists of the following components:

- Mediators that allow producers and consumers to discover each other and establish remote communications,
- Consumers typically subscribe to receive information or events from producers,
- Producers gather and publish information for consumers.

In Tycho, producers and/or consumers (clients) can publish their existence in a directory service known as the Virtual Registry (VR). A client uses the VR to locate other clients, which will act as a source or sink for the data they are interested in. The VR is a distributed service provided by a network of mediators. When possible, clients communicate directly, however for clients that do not have direct access to the Internet, the mediator provides wide-area connectivity by acting as a gateway or proxy into a localised Tycho installation. The messaging layer and its performance compared to that of Naradabrokering is described in more detail in [5]. Figure 1 shows Tycho clients communicating between two remote sites connected via the Internet. You can also see local clients communicating directly, bypassing the mediators.

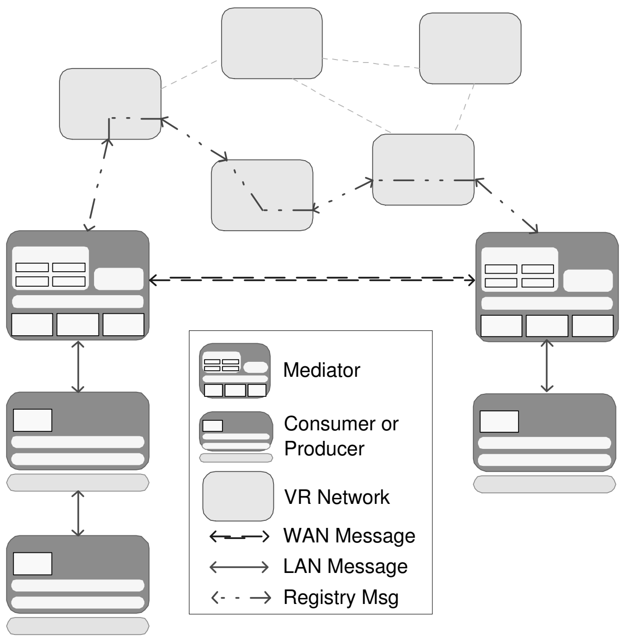

Figure 1. Tycho consumers and producers communicating over the Internet.

3.1 The Tycho VR

The Tycho VR is made up of a collection of pluggable services that provide for the management of client information and facilitate locating and querying remote Tycho installations. Clients register with the VR component within a local mediator when they start-up. The VR provides a locally unique name for each client and periodically checks registered clients to ensure their liveliness, removing stale entries if necessary.

The VR consists of the following components:

- The **transport handler** allows different transport protocols to be used between Tycho producers, consumers, mediators, and the VR. Currently the transport handler supports TCP sockets, HTTP (partly provided using an embedded instance of Jetty [11]) and Internet Relay Chat (IRC). Transport handler functionality can be extended by effectively creating a driver for each new protocol. It is possible to have more than one transport handler running at a time, as protocol descriptions are encoded in the URL of each end-point, which allows the selection of the appropriate transport handler for each protocol in use. For example, to add support for SOAP or HTTPS, an appropriate driver would be developed (or retrieved from a driver library) and plugged into the Tycho framework.
- The **local store** provides an abstract interface to a mediator's information store. The store itself can be implemented using a variety of data storage technologies. Currently Tycho provides a JDBC-based storage medium and an in-memory data structure (simple store). JDBC permits the use of a range of SQL storage technologies ranging from Oracle to MySQL. The in-memory implementation is provided to simplify the deployment of Tycho in situations where a JDBC-based database is unavailable. The JDBC store has been configured to make use of indexes to improve performance; the simple store takes a more naive approach, which will be appropriate for small numbers of records. The local store could also be implemented using LDAP [12], or RDF [13], depending upon a site's particular requirements.
- The **query parser** and **result annotator** components translate queries and responses into an intermediate internal format in order to allow Tycho to support different query languages and permit interoperability with other systems in the future. Tycho currently supports a subset of the ANSI-SQL query language and LDIF [14] as a response mark up.

When a mediator receives a query from a client, it performs a look up against locally registered entities, and then potentially dispatches the query to the rest of the VR. The local mediator requires a bootstrap service for locating other mediators and a transport handler for dispatching queries, which must be tolerant of mediator failures. Together, these services are provided by what we call the VR-interconnect, of which there are currently two:

- **HTTP:** This interconnect uses a well-known server to maintain a list of existing mediators within the VR, which it uses to bootstrap a new mediator. Inter-mediator communication is provided by the HTTP transport handler, which dispatches queries to remote mediators by sending messages serially to all mediators. This approach is not expected to be fault tolerant or the most scalable, but we expect good performance when transporting large queries as the transport handler has previously been optimised during earlier work. Currently access to the HTTP-interconnect can be restricted by placing Tycho behind a proxy server configured to require authentication.

- **IRC:** This interconnect uses a dynamic discovery process based on Internet Relay Chat (IRC); the service is discussed in detail in Section 3.2.

Tycho's VR provides:

- Information for uniquely identifying a client,

- URLs that are used by the transport handlers to locate and communicate with a client,

- A schema field, which can be used to store information about the capabilities of a producer or consumer. We expect this field to typically contain an XML document that can be searched through as part of a normal query. This approach provides greater functionality than other similar registries that typically use a single registry schema for all client records, for example MDS4. This approach allows Tycho to support a range of application's needs simultaneously within a VR without having to rely on manual configuration changes.

3.2 IRC VR-Interconnect

IRC networks [15] typically have groups of servers connected in a graph topology, which can be configured to route messages and provide fault tolerant capabilities. For example, QuakeNet, can support many hundreds of thousands of clients simultaneously [16]. The IRC DNS servers can be bound to a pool of IRC servers to provide load balancing based on server load and geographic location. A DNS query will then respond with the address of a 'suitable' lightly loaded server. Alternatively, by using a database of IP address prefixes, it can provide the address of a server that is geographically close to the client. In the event that a server becomes unavailable, DNS can be used to direct a client to available servers.

Tycho uses the combination of DNS records and the IRC servers to bootstrap the VR. This provides the VR service with a measure of fault tolerance, as it avoids a single point of failure, provides scalability and importantly by-passes the need to install servers to provide the functionality required by Tycho. An IRC client (bot) is then used by the VR to locate and communicate with other instances of the registry within the VR. Figure 2 illustrates this process:

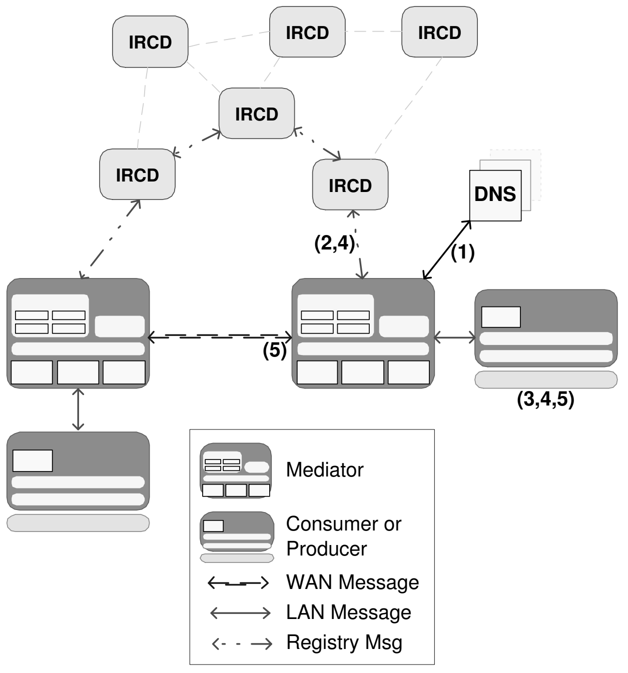

Figure 2. The steps a consumer takes to discover a producer using the IRC VR.

1. When the mediator is started, an IRC bot uses DNS information to locate a server and joins the IRC network.

2. The IRC bot attempts to join a pre-determined IRC channel.

3. A consumer will register itself with its local mediator.

4. (a) The consumer sends a query for producers to its local mediator.
   (b) The IRC bot within the local mediator sends the query to the IRC server, which broadcasts it to other bots associated with remote mediators.

(c) The query is run against the local data store at each mediator and matches are sent back over IRC and delivered to the consumer.

5. The consumer communicates with the producer via the mediator, in this case using a combination of sockets and HTTP.

IRC servers can be configured to use encryption to protect messages while in transit over the Internet. If a public IRC network is used for Tycho there are various services provided to help prevent unauthorised access to the VR bots. The channel through which the bots communicate can be password protected and bots can communicate using 'private messages', which the IRC network does not expose to other parties. Higher levels of security can be provided with a private IRC network, which allows the maximum amount of control over the security of the system, although it also adds the administrative overhead of maintaining an IRC network.

4 Performance Evaluation

The aims of the performance evaluation are firstly to compare the different implementations of both the Tycho local stores and the VR-interconnects in order to assess which offers the best performance in different situations. Secondly to measure the performance of the registries under load (with multiple simultaneous clients) and how they perform with increasing numbers of records in the registries.

We test Tycho against R-GMA and MDS4 in order to show that our philosophy of keeping the core functionality as simple as possible yields performance gains over other systems while still supporting the registry functionality required. In Tycho, more complex functionality is added to the edge of the implementation rather than by increasing the complexity of the core, thus is it is essential that the core perform well.

4.1 Configuration

Tycho, MDS4 and R-GMA all use different terminology to describe the same functional components. In the following sections we use the label 'registry' to refer the collection of services each system uses to provide registry functionality. For Tycho this is the mediator, for MDS4, the container running the MDS4 services and for R-GMA the Tomcat container running the registry and schema servlets. We call programs interacting with the registry 'clients'.

We generated a set of randomly generated strings to act as attributes for records to be inserted into the registries for the tests. A single record, with no mark up, had an average size of 114 bytes.

Two different queries are used to test the registry performance. The first **[S1]** simulates a client searching the registry for records matching some known attributes. Systematic queries are generated using a function to select a record name at random from the test data to guarantee the query will only match one record. In Tycho this is specified using the following SQL: `SELECT * FROM clients WHERE name='randname';`

The second query **[S2]** measures the worst case scenario of the client requesting all of the records from within the registry. This type of query will highlight the effect on client to registry communication, as the message response size increases with the number of records in the registry. In Tycho this is specified using the following SQL: `SELECT * FROM clients;`

By configuring the Tycho core VR services, described in Section 3.1, and arranging these components in different ways we have been able to test the performance of the Tycho's VR under variety of different circumstances and compare it to the performance or MDS4 and R-GMA. In Section 4.2, we outline the measurements of interest and how the components were arranged to perform each test.

4.2 Methodology

A nine-node cluster was used to perform the performance tests. Each node has dual 2.8 GHz Xeon processors connected by Fast Ethernet with 2 Gbytes of RAM. The cluster uses Debian Linux 3.1, with the 2.4.32 kernel. Java 5 was used for Tycho and MDS4 (Sun's JVM version 1.5.0-b64), R-GMA requires Sun JVM 1.4.2_06. Software versions used were Tycho 0.7.3, MDS4 from Globus 4.0.1 and R-GMA from gLite 1.5. All three systems were run with their security features disabled.

Figure 3 shows how the components were arranged on the cluster for the tests. Node 0 is the cluster's head node, and node 1 to N are the other nodes.

Test 1 - Local stores: In this test the number of producers registered in a single mediator on one cluster node was varied from 10 to 100,000. A Tycho client on a second cluster node queried the VR using the two different SQL queries (S1 and S2). The test aims to measures the performance of the different back-end stores to show which one performs best under different circumstances. The test was repeated for each different store. The time to complete each query was recorded.

Test 2 - VR-interconnects: In this test a single Tycho client on one node queried the whole VR for a single record. The VR was made up of 1-1000 mediators evenly distributed on the compute nodes. Each mediator contained 1000 records so as the number of mediators increased so did the total size of the VR (reaching a maximum of 100,000 total records when using 1000 mediators). This test measures

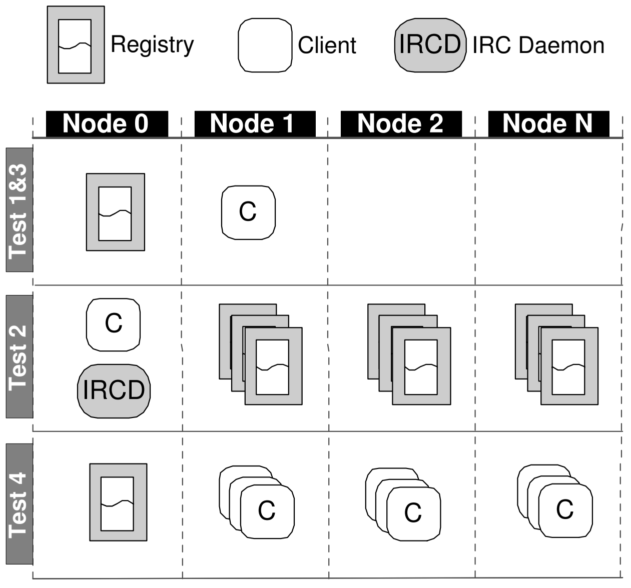

Figure 3. Test configurations.

the performance of the Tycho VR as the number of mediators in the system is increased. We tested the two different VR interconnect protocols, IRC and HTTP, with caching on and off. When testing IRC we used an open source IRC daemon called ngIRCd [17]. The average response time for a query was measured.

Test 3 - Records: We repeated test 1 using R-GMA and MDS4, we used Tycho with the HSQLDB store for comparison because it currently provides the best results for an in memory store. In this test the number of records published into a single registry on one cluster node was varied from 10 to 100,000. A client on a second cluster node queried the registry using the two different types of queries (S1 and S2). The test aims to measure the performance of the different registries with increasing numbers of records. The time to complete each query was recorded.

Test 4 - Clients: In this test the three registries were loaded with 1000 records and the number of clients performing simultaneous queries was varied from 1-1000. The registry was run on one node and the clients were evenly distributed amongst the eight other nodes. This test measured the effect of increasing the number of simultaneous queries on a single registry; it attempts to show how well a single registry copes with increasing numbers of local clients. The time to complete a query was measured.

The tests were managed with shell scripts, which used SSH to execute the correct test components on each machine. In tests 2 and 4 the clients and registries were evenly distributed using a round-robin approach between the compute nodes. So for example, if testing 64 clients, each compute node had 8 clients. The cluster was monitored using Ganglia to provide information such as memory usage. Each test was repeated 10,000 times and the benchmarks were written so that the time to bootstrap the test did not interfere with the measurements being gathered. For Tycho, the JVM was invoked with no command line options. With MDS4 and R-GMA, when the default heap size was exhausted the tests were repeated using up to a maximum heap of size 1.5 Gbytes. The Java timer used in the tests had microsecond resolution. All the times shown are in milliseconds (ms).

4.3 Analysis of Test Results

4.3.1 Test 1 (Tycho)

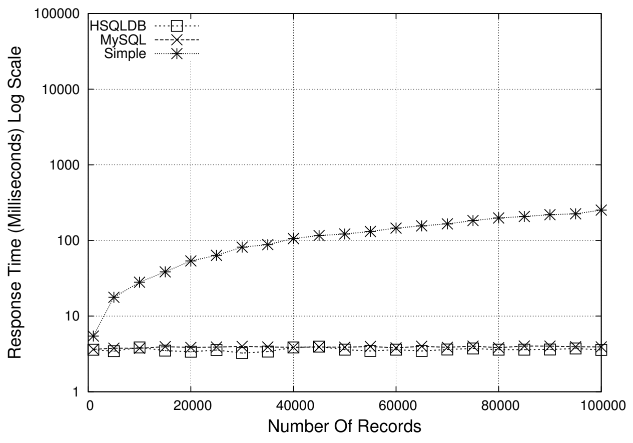

Figure 4. Response time (ms) versus the number of records using different data stores for queries that select a single random record.

When a single random record is retrieved from a store (Figure 4) the response time remains constant at around 3.5 ms for both HSQLDB and MySQL. This is because they are both able to hash directly (see Section 3.1) to the required record using indexing. Whereas the Simple store iteratively checks every record, as it does not use indexing.

When selecting all of the records (Figure 5) the performance for all three stores are comparable until around 5000 records. At 100,000 records, (a 10.59 Mbyte response) the Simple store is 1550 ms faster than MySQL and 1350 ms faster than HSQLDB. The Simple store is fastest because it does not have the overhead of using JDBC to retrieve the records. The Simple store at 70,000 records (a 8.45 Mbyte response) shows a jump of 2000 ms. The exact cause of this is unknown, but we suspect it is due to the increasing size of the Java String objects within the store, and the effect of garbage collector attempting to recover memory.

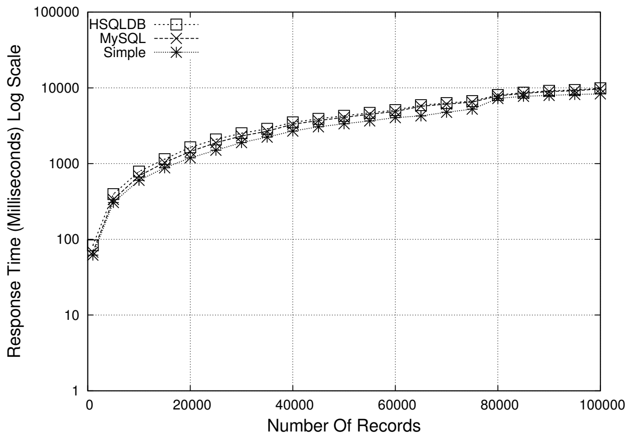

Figure 5. Response Time (ms) versus the number of records using different data stores for queries that select all records.

4.3.2 Test 2 (Tycho)

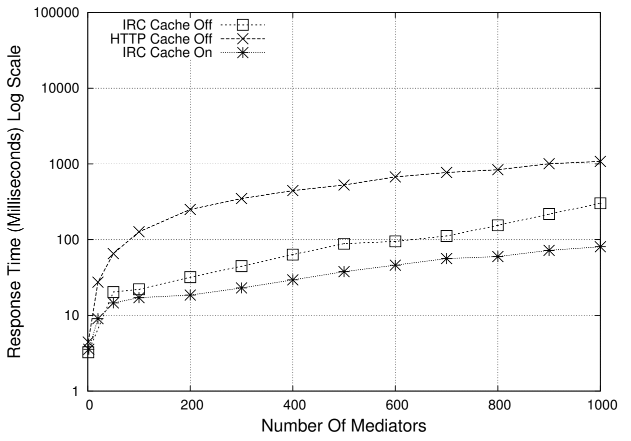

Figure 6. Response time (ms) versus the number of mediators.

Figure 6 shows the effect on response time as the number of mediators and records in the VR is increased. For HTTP (no caching) an extra 1.24 ms is added per new mediator. This performance is attributed to the serial way the HTTP dispatches queries to other mediators. With IRC (no caching) until 500 mediators (50,0000 records) each extra mediator and 1000 records add on average 0.41 ms to the response time. The peak at 500 mediators for IRC (no caching) marks the point where the test cluster consumed its available RAM and started to use swap space. The marked difference between the performance of IRC and HTTP is mainly because queries are sent to the mediators in parallel by the IRC daemon as opposed to serially for HTTP. IRC (caching on) has the lowest average response time adding approximately 0.13 ms for each extra mediator.

4.3.3 Test 3 (Tycho, R-GMA and MDS4)

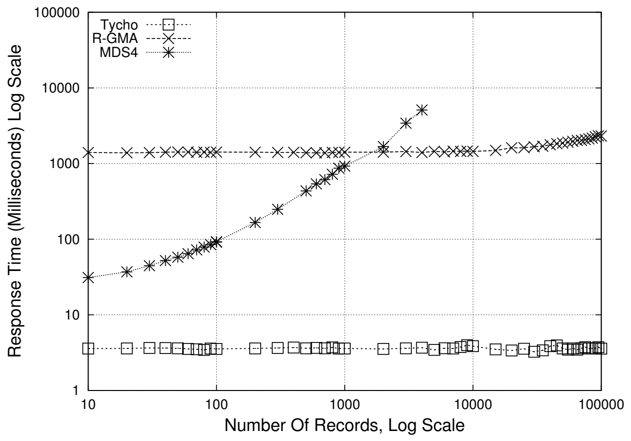

Figure 7. Response time (ms) versus the number of records for queries that select a single random record.

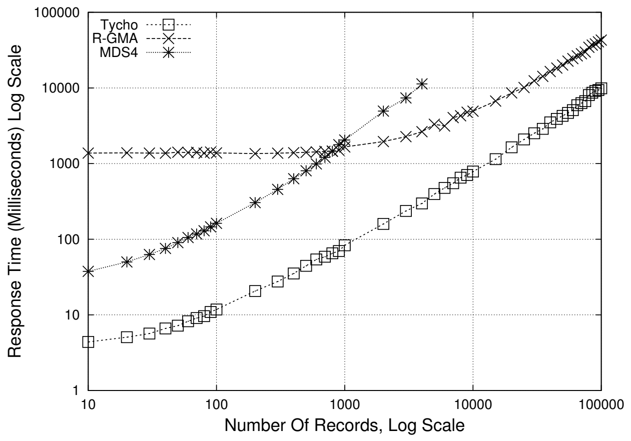

Figure 8. Response Time (ms) versus the number of records for queries that select all records.

Figures 7 and 8 shows the effect on response time when increasing the number of records in the registry of Tycho, R-GMA and MDS4. In Figure 7 a single random record is being selected. Tycho has a constant response time of 3.5

ms, this is consistent with Test 1 which showed that for up to 100,000 records HSQLDB has a constant overhead for this simple query. R-GMA has a fixed query time of 1400 ms up to 20,000 records at which point it increases steadily to 2314 ms for 100,000 records. Unlike R-GMA and Tycho, MDS4 response time increases with every extra record. It starts with a query time of 21.06 ms for 10 records and increases steadily to 5095 ms for 4000 records. After 4000 records the Globus container executing the MDS4 services gave an out of heap error with the heap size set to the maximum supported by the test hardware of 1.5 Gbytes.

Figure 8 uses the same test data as the previous test, but the query being run selects all of the records in the index. Both Tycho and MDS4 have a constant increase in response time in relation to the number of records with Tycho increasing by around 0.08 ms per record and MDS4 1.93 ms. R-GMA has a constant response time of approximately 1462 ms for up to 2000 records, this can probably be attributed to some kind of fixed cost in the implementation. After this point the response time increases steadily up to 42,871 ms for 100,000 records, which is 32,973 ms more than Tycho.

It is interesting to note that the curves for MDS4, for both types of query, track each other closely. The difference increases from 71.6 ms for 100 records, up to 6208 ms for 4000 records. We believe this is because the network latency has a greater impact for the larger response size associated with the select all query.

4.3.4 Test 4 (Tycho, R-GMA and MDS4)

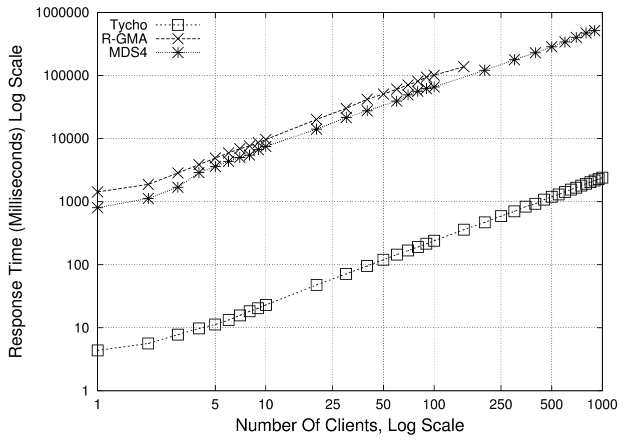

Figure 9. Response time (ms) versus the number of clients concurrently querying a single mediator.

Figure 9 shows the relationship between the query response time as the number of concurrent clients that query a single registry. increases. For Tycho the response time increases by on average 2.3 ms per additional client. The response time for MDS4 is on average a extra 14,077 ms higher than Tycho, and for each additional client approximately 655 ms is added. R-GMA has an additional 20,094 ms latency than Tycho for the same number of clients (6016 ms higher than MDS4) with an increase of 989 ms to the response time per concurrent client. When testing R-GMA, after 150 clients the registry servlet crashed with an out of stack memory error preventing us from completing the tests up to 1000 clients. We believe this is due to the rapid creation and destruction of the R-GMA consumers, the test code selects a random record per iteration and in R-GMA a new consumer must be created to run a different query.

5 Summary and Conclusions

In this paper, we have described our motivation for developing Tycho with its combined wide-area messaging framework and built-in distributed registry (VR). We outlined the architecture of Tycho, which allows it provide a range of services and explained the design and functionality of the VR. The results of benchmarks, comparing Tycho's registry with R-GMA and MDS4, as well as more extensive tests have been presented and discussed.

Tests 1 and 2 (Section 4.3.1 and 4.3.2) measured the performance of the different component implementations currently supported by the Tycho VR in a range of configurations. The Simple store performed best for small numbers of records, but HSQDL and MySQL scale better with number of records. The Simple store could be improved by making the internal search mechanism more intelligent, for example using hashing.

The IRC interconnect performed better when routing queries between mediators than HTTP. For 1000 mediators, the response time was 778 ms faster, and with caching the latency was reduced by a further 220 ms. The HTTP interconnect could be improved by sending queries to mediators in parallel and the IRC-interconnect could be further improved by using multiple IRC channels to provide a network overlay to allow more efficient message routing. The HTTP-interconnect can perform better than IRC as message response size increases, this is a result of the IRC having to split the response into multiple messages due to limitations of the IRC protocol. One solution to overcome these issues would be to use a hybrid VR-interconnect using a combination of transport handlers to exploit the strengths of the different approaches, i.e. using IRC to route queries and HTTP to deliver responses over a certain size.

When testing the effect of number of records on response size (Section 4.3.3 we see that when selecting a single record from 100,000, Tycho responds 32 seconds faster than R-GMA. MDS4 runs out of heap space for larger records sizes, which suggests that they should look at either storing the data more efficiently or moving to a file backed store that is not limited by heap size. The results of testing the Tycho stores using HSQLDB, MySQL and the Simple Java store show that HSQLDB performs best, perhaps R-GMA should consider using this store instead MySQL too.

In the multiple client tests (Section 4.3.4) Tycho's VR had a lower response latency than R-GMA and MDS4. With 100 clients Tycho was 94 seconds faster than R-GMA and 65 seconds faster than MDS4. The results highlight that one of the strengths of our implementation is its performance under load. Tycho's performance is linear with regard to both increasing numbers of clients and response sizes.

MDS4 also uses a global schema which must be consistent in every MDS4 instance for interoperability, this reduces its flexibility. R-GMA is the closest match to Tycho's architecture although it does not allow for the same level of flexibility with regard to the data it can store as the schema is global to the R-GMA system and must be configured before records can be inserted.

As mentioned in Section 3, Tycho can leverage the security features of existing tools such as those provided by the IRC daemon or a HTTP proxy to prevent unauthorised access to the VR. In keeping with the design philosophy of Tycho we are looking for ways to incorporate established security mechanisms into the VR to provide more advanced security features, one possible route is to provide a service to provide WS-Security compatible functionality.

5.1 Tycho's Use in Other Systems

Tycho is being used to provide service discovery for the VOTechBroker [18], which is part of the European Virtual Observatory [19] project. The Virtual Observatory will allow astronomers global access via a web portal to various astronomical data archives. The VOTechBroker facilitates the use of existing infrastructure to execute jobs submitted through a web interface. Participating sites publish capabilities, such as the batch submission details via a Tycho producer; the VOTechBroker uses a Tycho consumer to discover the remote resources and uses the capabilities published via Tycho to select a site to submit the jobs. We expect the process of integrating Tycho with other systems such as the VOTechBroker will lead to the development of more sophisticated tools and services, such as aggregate and multi threaded producers.

Tycho [20] is currently available as a binary release to developers interested in investigating and further enhancing its capabilities.

5.2 Future Work

Even though Tycho's registry performance is better then both MDS4 and R-GMA, there are still an number of areas that we feel could be improved.

One way to improve performance is altering caching in the mediator to include local data-store queries in addition to remote responses. Adding indexing to the Simple store would improve its performance when searching for records. In addition, the message-passing performance could be improved by changing the socket transport handler to use thread pooling to further reduce the cost of sending messages.

In the future, we will add functionality into Tycho to provide services that are more advanced. One key area is to develop transport handlers that support SSL sockets or HTTPS to provide secure communication. Other features may include support for transactions, various Web Services specifications, for example WS-notification, and producers/consumers that are suitable for computational steering.

Acknowledgements

This research was supported in part by the Grid Technology Group of the CCLRC e-Science Centre, Daresbury Laboratory, UK.

References

[1] SOAP Version 1.2 Part 1: Messaging Framework, June, 2003, http://www.w3.org/TR/soap12-part1/, W3C Recommendation.

[2] The Globus Toolkit, GridFTP, accessed 07 July 2006, http://www.globus.org/toolkit/docs/4.0/data/gridftp/.

[3] OASIS, UDDI Version 2.03 Replication Specification, July, 2002, UDDI Committee Specification.

[4] J. Schopf, I. Raicu, L. Pearlman, N. Miller, C. Kesselman, I. Foster and M. D'Arc, Monitoring and Discovery in a Web Services Framework: Functionality and Performance of Globus Toolkit MDS4, January, 2006, Argonne National Laboratory.

[5] M.A. Baker, M. Grove and R. Lakhoo, A Preliminary Performance Evaluation of jGMA With the NaradaBrokering Framework, Technical Report, June 2005.

[6] S. Pallickara and G. Fox, NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids, Proceedings of

ACM/IFIP/USENIX International Middleware Conference Middleware-2003. pp 41-61, Lecture Notes in Computer Science 2672 Springer 2003, ISBN 3-540-40317-5.

[7] A.W. Cooke, et al, The Relational Grid Monitoring Architecture: Mediating Information about the Grid, Journal of Grid Computing, 2, 323-339, 2004.

[8] The DataGrid Project, accessed 30 May 2006, http://eu-datagrid.web.cern.ch/eu-datagrid/.

[9] I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, 2005, IFIP International Conference on Network and Parallel Computing, Springer-Verlag.

[10] OASIS, Web Services Resource Framework (WSRF) - Primer 1.2, May, 2006.

[11] Jetty, accessed 30 May 2006, http://jetty.mortbay.org/jetty/.

[12] OpenLDAP, accessed 07 July 2006, http://www.openldap.org/.

[13] Resource Description Framework (RDF), accessed 30 May 2006, http://www.w3.org/RDF/.

[14] G. Good, RFC 2849 - The LDAP Data Interchange Format (LDIF) - Technical Specification, June, 2000, http://www.faqs.org/rfcs/rfc2849.html, RFC.

[15] J. Oikarinen and D. Reed, RFC 1459 - Internet Relay Chat Protocol, May, 1993, http://www.faqs.org/rfcs/rfc1459.html, RFC.

[16] irc.netsplit.de - IRC search engine and statistics, accessed 07 July 2006, http://irc.netsplit.de/networks/.

[17] ngIRCd: Next Generation IRC Daemon, accessed 07 July 2006, http://ngircd.barton.de/.

[18] The VOTechBroker (VOTB) Project, accessed 07 July 2006, http://dsg.port.ac.uk/projects/votb/

[19] European Virtual Observatory, accessed 07 July 2006, http://euro-vo.org/.

[20] Tycho - A Wide-area Distributed Messaging Framework, accessed 07 July 2006, http://dsg.port.ac.uk/projects/tycho/.