An Information Architecture Based on Publish/Subscribe Messaging

Warren Smith

Texas Advanced Computing Center
University of Texas at Austin
Overview

• Motivation
• Architecture
• Performance
• Prototype
• Conclusions & Future Work
Motivation

• Providing advance scheduling tools for TeraGrid
  – Condor-G matchmaking (automatic resource selection)
  – Wait time predictions for jobs in batch queues
• Needed more information than was available
  – Created GLUE 2 information providers
    • Information model and bindings defined by the OGF
  – Information distributed via TeraGrid IIS
• GLUE 2 information received from IIS was 10 to 20 minutes old
  – Making decisions or predictions based on old information
  – Impacts performance and accuracy!
• Want to provide more timely information to tools
Querying for Information

- Querying (or searching) typically used to retrieve information

- If updates are desired, how often to query?
  - Query too often -> excess load on the information service
    - Even worse if there are many clients doing this (e.g. a popular per-user tool)
  - Query less often -> tool operating on old information
    - Less efficient decisions
Query-Based Information Architecture

- Situation is worse in some cases
  - Periodic updates to local information stores
  - Periodic query/response to update centralized information store
  - Even multiple central information stores with periodic updates
- Periodic updates add up
  - 3 updates, 5 mins each -> 15 mins worst case
- This was the situation in TeraGrid
Publish/Subscribe Messaging

- Centralized messaging service
  - Not always required, but good for our situation
- Publisher publishes messages to service
  - Doesn’t know who is interested in them
- Subscriber subscribes to service for messages
  - Based on topic, content, …
  - Doesn’t know who is publishing
- Service routes messages to subscribers
  - Matching to subscriptions
- Often a push/notification model
  - Service handles messages as they arrive
  - Subscriber gets an immediate callback when a message arrives
Performance Experiments

• Message broker
  – RabbitMQ 2.4.1
    • Advanced Message Queuing Protocol (AMQP)
      – Popular implementation
    • Responsive email support
      – Commercial support available
    • Active user community
    • Open source
    • Erlang

• Clients
  – RabbitMQ Java
  – Pika (Python)
Experimental Environment

- **FutureGrid**
  - Distributed experimental test-bed
- **Service on one cluster, clients on another**
  - Software deployed on base Linux OS (no virtualization)
- **Compute nodes**
  - Dual socket Nehalem
  - 32 GB memory
- **10 Gb/s network**
1 to 1 Message Throughput

![Graph showing 1 to 1 Message Throughput with two lines representing Java Client and Python Client]
1 to 1 Message Bandwidth

Bandwidth (MB/s)

Message Size (bytes)
1 to 1 Latency

Latency (seconds)

Message Size (bytes)

Java Client

Python Client
Latency – Many Subscribers

Latency (seconds)

Message Size (bytes)

1 subscriber
8 subscribers
64 subscribers
512 subscribers
## TeraGrid Information Providers

<table>
<thead>
<tr>
<th>Information Type</th>
<th>Document Sizes (bytes)</th>
<th>Median Document Size (bytes)</th>
<th>Update Frequency (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability Deployment</td>
<td>10K</td>
<td>111K</td>
<td>26K</td>
</tr>
<tr>
<td>Local Software</td>
<td>1K</td>
<td>644K</td>
<td>37K</td>
</tr>
<tr>
<td>GLUE 2 Resources</td>
<td>8K</td>
<td>69K</td>
<td>13K</td>
</tr>
<tr>
<td>GLUE 2 Jobs</td>
<td>345</td>
<td>3.3M</td>
<td>339K</td>
</tr>
<tr>
<td>User Portal Load</td>
<td>208</td>
<td>310</td>
<td>294</td>
</tr>
<tr>
<td>User Portal Jobs</td>
<td>142</td>
<td>2.8M</td>
<td>45K</td>
</tr>
</tbody>
</table>

- Above is most of the dynamic info in TeraGrid Integrated Information Services
  - Doesn’t include accounting information
TeraGrid Consumers

• Centralized information services
  – WS-MDS
  – REST
• User portal
• Metascheduling tools
  – Condor-G matchmaking
  – Karnak queue predictions
• Science Gateways
TeraGrid Emulation

• Current
  – 20 TeraGrid resources that could publish
    • They aren’t all publishing all of the info
  – Approximately 10 consumers
    • Assume they want all published info (they don’t)

• In XSEDE
  – More resources
    • Campus bridging
  – More consumers
    • Assuming people like pub/sub
  – More information?
    • Services (job management, file transfer, accounting, …)
    • Science Gateways publishing
Emulation Results

Latency (seconds)

Message Size (bytes)

Increase in messages

- 20 publishers, 10 subscribers: 23x
- 50 publishers, 100 subscribers: 228x
- 100 publishers, 500 subscribers: 455x
- 100 publishers, 1000 subscribers: 2.3

0.95

288
Prototype

- Deploy prototype to validate approach
- Initial prototyping on FutureGrid
  - Using virtual machines to easily try/organize different software configurations
- TeraGrid Prototype
  - Deploy on TeraGrid servers and resources
  - Integrate with existing information services
  - Worked well
  - Ran out of time to harden and deploy in production before XSEDE
Prototype Messaging Service

- RabbitMQ broker
- Two instances clustered into a single broker
  - Subscribers to one receive messages sent to the other
  - Basic fault tolerance
    - Clients know to try one service if the other is down
  - May also improve performance
- Clients authenticate using X.509 certificates
- Authorization based on Distinguished Names in certificates
- Anonymous users can receive non-sensitive information
Prototype Publishing

• TeraGrid GLUE 2 implementation
  – Document with compute resource descriptions
  – Document with job information
• Deployed on compute resources
• Modified to publish messages
  – Using Pika Python client
  – X.509 host certificate for authn/authz
• Runs periodically out of cron
Prototype Bridging

- IIS Documents -> messages
  - Make current information available as messages
- Messages -> IIS documents
  - Make new information available via current information services
- Translate GLUE 2 documents to user portal documents and publish
  - SPs won’t need to run the User Portal Providers
  - Or the globus-wsrf container, if SP isn’t running GRAM 4
- Tricky to configure all the above correctly
- Currently one service
  - Plan services running on each of the two IIS servers
    - Use messages to coordinate activity
Conclusions

• Publish/subscribe approach is promising
• Production-quality software
  – Installation and configuration isn’t hard
  – Excellent performance for our needs
  – Compatible authentication mechanisms
  – Flexible authorization mechanisms
• Appropriate for a number of uses
  – When user/tool wants updates
    • Information arrives very quickly (pushed)
  – Publishing is easy
    • Cron jobs – don’t need to run a service
• Good basis for query-based information services
  – Information can arrive at such services via messages
• Integrates well with current TeraGrid IIS
Future Work

• Include databases in the evaluation
  – XML, SQL?, NoSQL?

• XSEDE will be deciding how to redesign TeraGrid information services
  – This is one possible approach

• Handle new XSEDE use cases
  – Many TBD
  – Integration with non-XSEDE information sources
    • Open Science Grid
    • Campus resources