Scalable, Fault-tolerant Management of Grid Services

Harshawardhan Gadgil, Geoffrey Fox, Shrideep Pallickara, Marlon Pierce

Presented By

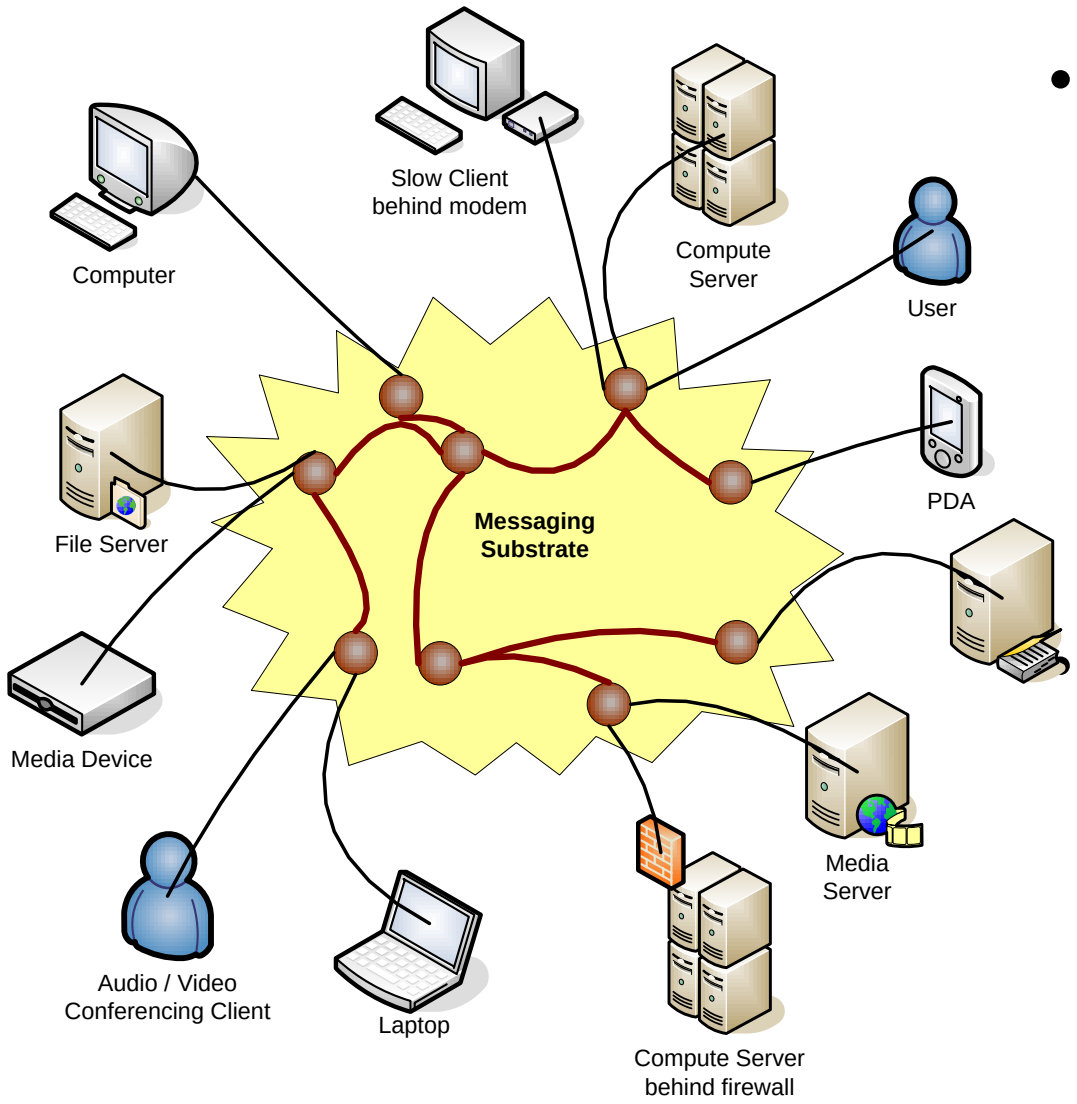
Harshawardhan Gadgil (hgadgil@cs.indiana.edu)

Talk Outline

- Use Cases and Motivation
- Architecture
  - Handling Consistency and Security Issues
- Performance Evaluation
- Conclusion
  - Contributions and Future Work

Grid

Large Number of Distributed Resources



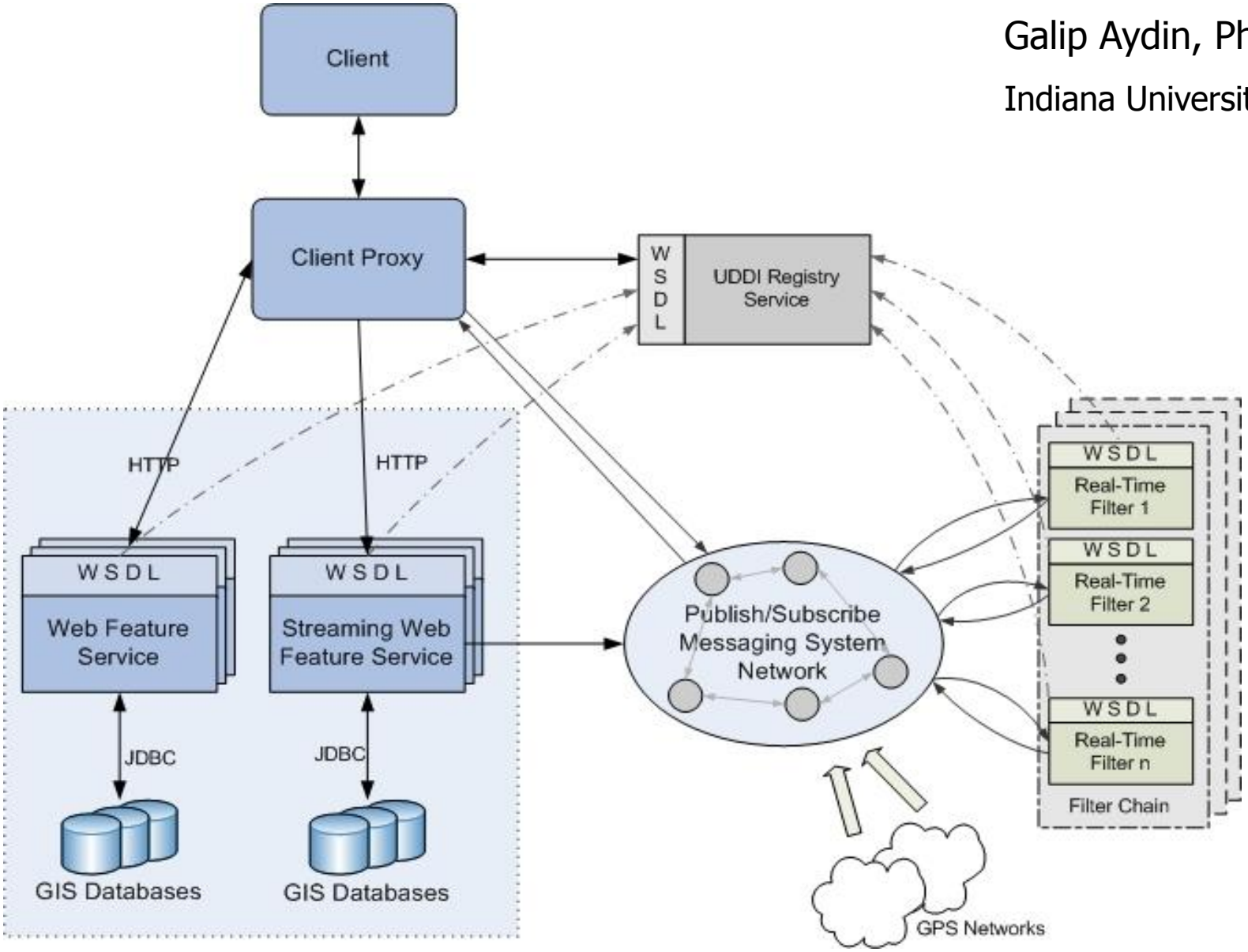
- Applications distributed and composed of a large number and type (hardware, software) of resources

- Components widely dispersed and disparate in nature and access

Sensor Grid*

Galip Aydin, Ph.D. Thesis

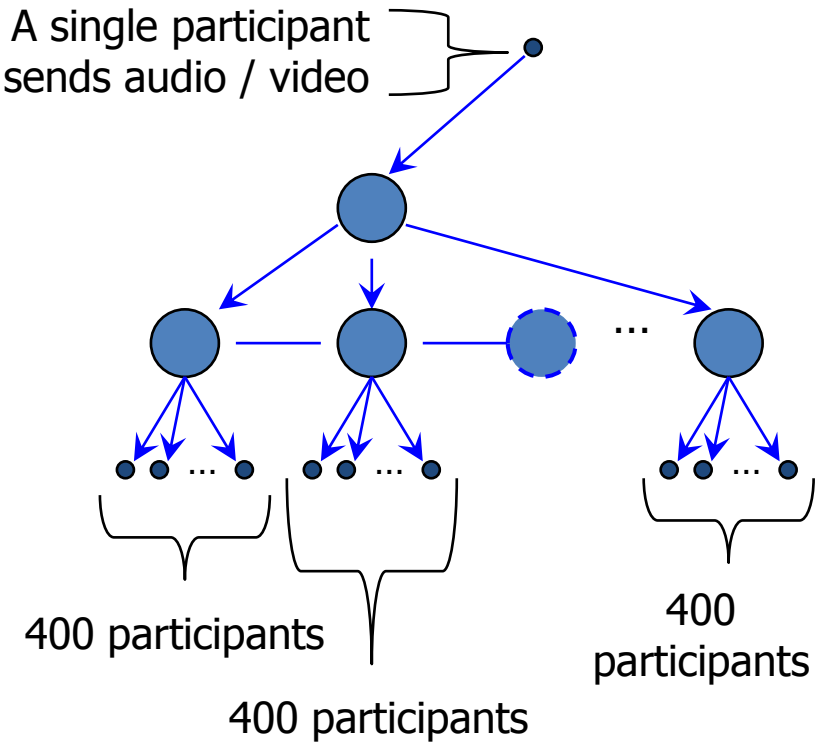
Indiana University, Jan 2007



Example

Audio Video Conferencing

- GlobalMMCS (http://www.globalmmcs.org) uses NaradaBrokering as an event delivery substrate
- Consider a scenario where there is a teacher and 10,000 students. One way is to form a TREE shaped hierarchy of brokers
- One broker can support up to 400 simultaneous video clients and 1500 simultaneous audio clients with acceptable quality*.
  - So one would need (≈ 10000 / 400 = 25 broker nodes).



* *"Scalable Service Oriented Architecture for Audio/Video Conferencing"*, Ahmet Uyar, Ph.D. Thesis, May 2005

Definition:

What is Management ?

- Service Management – Maintaining System's ability to provide its specified services with a prescribed QoS
- Management Operations* include
  - Configuration and Lifecycle operations (CREATE, DELETE)
  - Handle RUNTIME events
  - Monitor status and performance
  - Maintain system state (according to user defined criteria)
- This paper addresses:
  - Configuring, Deploying and Maintaining Valid Runtime Configuration
    - Crucial to successful working of applications
    - Static (configure and bootstrap) and Dynamic (monitoring / event handling)

***From** WS – Distributed Management

http://devresource.hp.com/drc/slide_presentations/wsdm/index.jsp

Existing Systems

- Distributed Monitoring frameworks
  - NWS, Ganglia, MonALISA
  - Primarily serve to gather metrics (which is one aspect of resource management, as we defined)
- Management Frameworks
  - SNMP – primarily for hardware (hubs, routers)
    - CMIP – Improved security & logging over SNMP
  - JMX – Managing and monitoring for Java applications
  - WBEM – System management to unify management of distributed computing environments
- Management systems not-interoperable – Move to Web Services based management of resources
  - XML based interactions that facilitate implementation in different languages, running on different platforms and over multiple transports
  - Competing Specifications (WS – Management and WS – Distributed Management)

Motivation:

Issues in Management

- Services must meet
  - General QoS and Life-cycle features
  - (User defined) Application specific criteria
  - Improper management such as wrong configuration – major cause of service downtime
- Large number of widely dispersed Services
  - Decreasing hardware cost => Easier to replicate for fault-tolerance (Espl. Software replication)
  - Presence of firewalls may restrict direct access to Services
- Service specific management systems have evolved independently (different platform / language / protocol)
  - Requires use of proprietary technologies
- Central management System
  - Scalability and single point of failure

Desired Features of the Management Framework

- Fault Tolerance
  - Failures are Normal, Services may fail, but so also components of the management framework.
  - Framework MUST recover from failure
- Scalability
  - With Growing Complexity of application, number of Services (application components) increase
    - E.g. LHC Grid consists of a large number of CPUs, disks and mass storage servers (on the order of ~ 30K)
  - In future, much larger systems will be built
  - MUST cope with large number of Services in terms of
    - Additional components Required

Desired Features of the Management Framework

- Performance
  - Initialization Cost, Recovery from failure, Responding to runtime events
- Interoperability
  - Service exist on different platforms, Written in different languages, managed using system specific protocols and hence not INTEROPERABLE
  - Framework must implement interoperable protocols such as based on Web-Service standards
- Generality
  - Management framework must be a generic framework
  - Should be applicable to any type of resource (hardware/ software). This paper primarily focuses on Service Management
- Usability
  - Autonomous operation (as much as possible)

Architecture

- Applicable to services which can be controlled by modest external state
- We assume Service specific external state to be maintained by a Registry (assumed scalable, fault-tolerant by known techniques)
- We leverage well-known strategies for providing
  - Fault-tolerance (E.g. Replication, periodic check-pointing, request-retry)
  - Fault-detection (E.g. Service heartbeats)
  - Scalability (E.g. hierarchical organization)

Management Architecture built in terms of

- Hierarchical Bootstrap System
  - Services in different domains can be managed with separate policies for each domain
  - Periodically spawns a System Health Check that ensures components are up and running

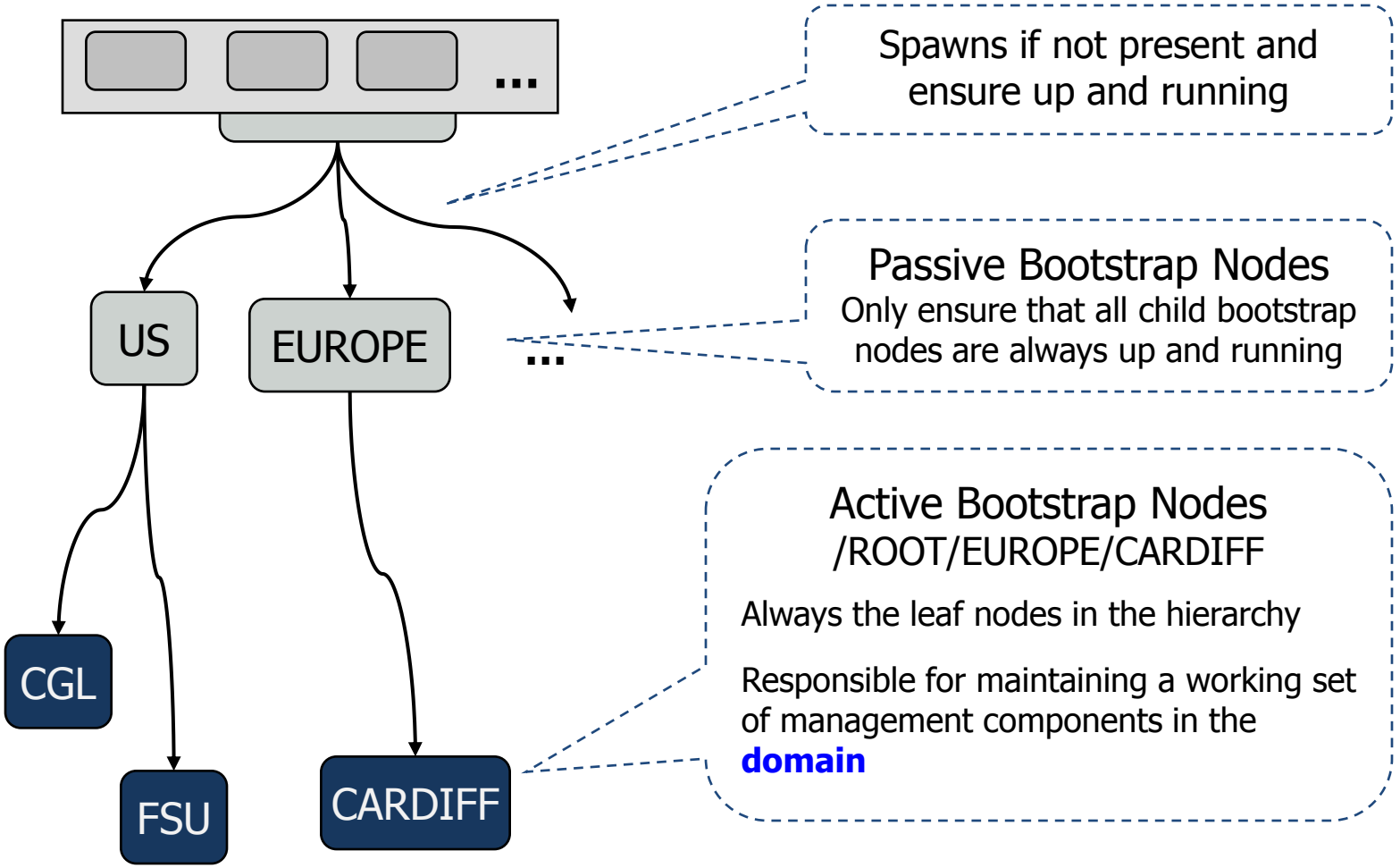
- Registry for metadata (distributed database) – Robust by standard database techniques and our system itself for Service Interfaces
  - Stores Service specific information (User-defined configuration / policies, external state required to properly manage a service)
  - Generates a unique ID per instance of registered component
  - Our present implementation is a simple registry service

Management Architecture built in terms of

- Messaging Nodes form a scalable messaging substrate
  - Provides transport protocol independent messaging between components
  - Can provide Secure delivery of messages
  - In our case, we use NaradaBrokering Broker as a messaging node **(http://www.naradabrokering.org)**
- Managers – Active stateless agents that manage services.
  - Since they don't maintain state, hence robust
  - Actual management functions are performed by a service specific manager component
- Services – what you are managing
  - Wrapped by a Service Adapter which provides a Web Service interface.
  - Service Adapter connects to messaging node to leverage transport independent publish subscribe communication with other components

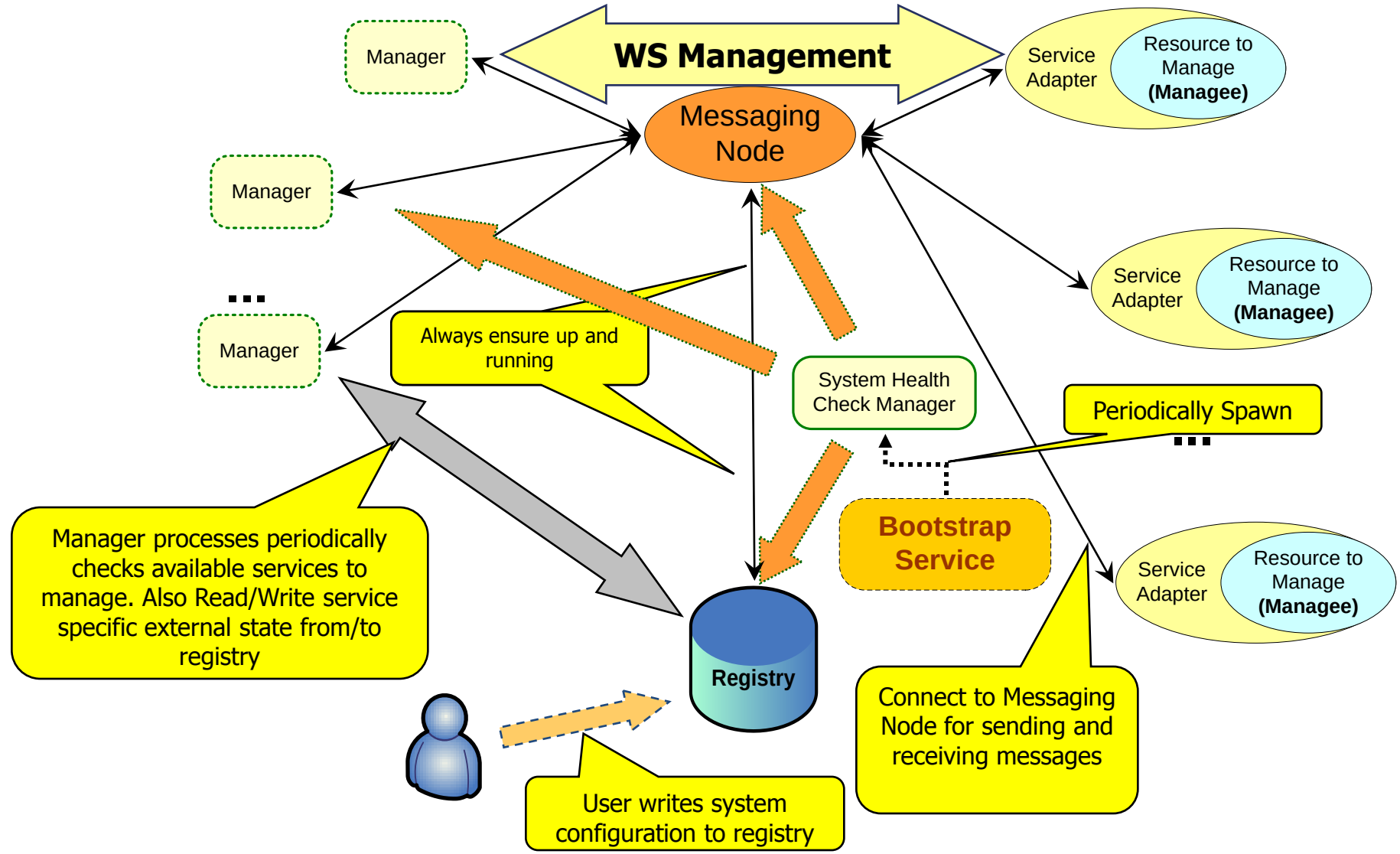
Architecture:

Scalability: Hierarchical distribution



Architecture:

Framework Components



↔ Publish Subscribe based communication via Messaging Node

Architecture

User Component

- Service Characteristics are determined by the user (Administrator for the services in question)
- Events generated by the services are handled by the manager
  - Event processing is determined by via WS-Policy constructs

    For e.g., Automatically instantiate a failed service instance

```
<pol:Policy xmlns:pol=http://schemas.xmlsoap.org/ws/2004/09/policy
  xmlns:pol1="http://www.hpsearch.org/schemas/2006/07/policy">
  <pol:All>
    <pol1:AUTOInstantiate
        forkProcessLocator="udp://156.56.104.152:65535"/>
  </pol:All>
</pol:Policy>
```

- Managers can set up services
  - A set of services can be started by simply writing appropriate info to Registry

Issues in the distributed system

Consistency

- Examples of inconsistent behavior
  - Two or more managers managing the same service
  - Old messages / requests reaching after new requests
  - Multiple copies of services existing at the same time / Orphan services leading to inconsistent system state
- Use a Registry generated monotonically increasing Unique Instance ID (IID) to distinguish between new and old instances
  - Requests from manager A are considered obsolete IF IID(A) < IID(B)
  - Service Adapter stores the last known MessageID (IID:seqNo) allowing it to differentiate between duplicates AND obsolete messages
  - Service adapter periodically renews with registry
    - IF IID(serviceInstance_1) < IID(serviceInstance_2)
    - THEN serviceInstance_1 is OBSOLETE
    - SO serviceInstance_1 silently shuts down

Issues in the distributed system

Security

- NaradaBrokering's Topic Creation and Discovery* and Security Scheme# addresses
  - Message level security
  - Provenance, Lifetime, Unique Topics
  - Secure Discovery of endpoints
  - Prevent unauthorized access to services
  - Prevent malicious users from modifying message
    - Thus message interactions are secure when passing through insecure intermediaries

* NB-Topic Creation and Discovery - Grid2005 / IJHPCN

NB-Security (Grid2006)

Implemented:

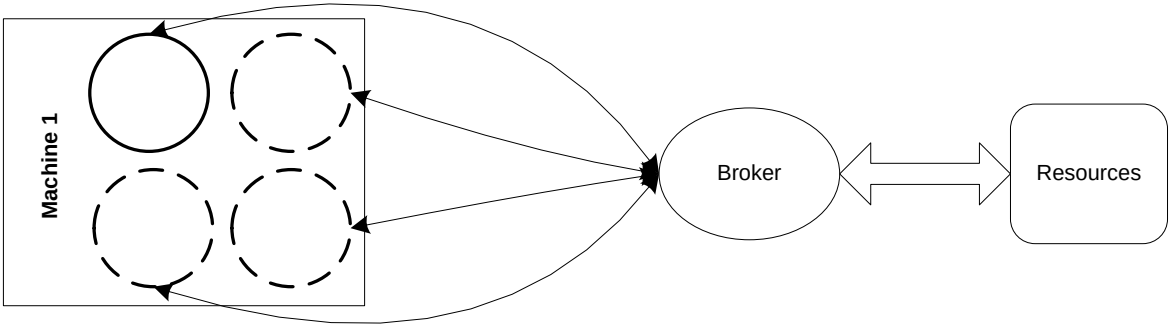
- Management framework
- WS – Specifications
  - WS – Management (could use WS-DM) -June 2005 parts (WS – Transfer [Sep 2004], WS – Enumeration [Sep 2004]) and WS – Policy[Sep 2004], SOAP v 1.2 (needed for WS-Management)
  - WS – Eventing (Leveraged from the WS – Eventing support in NaradaBrokering)
  - Used XmlBeans for manipulating XML in custom container
- Management of NaradaBrokering Brokers*
  - Released with NaradaBrokering in Feb 2007
  - Currently being used as a Grid builder tool to remotely deploy Grids dynamically (Rui Wang, Anabas.com)

*Managing Grid Messaging Middleware
Harshawardhan Gadgil et.al, CLADE 2006

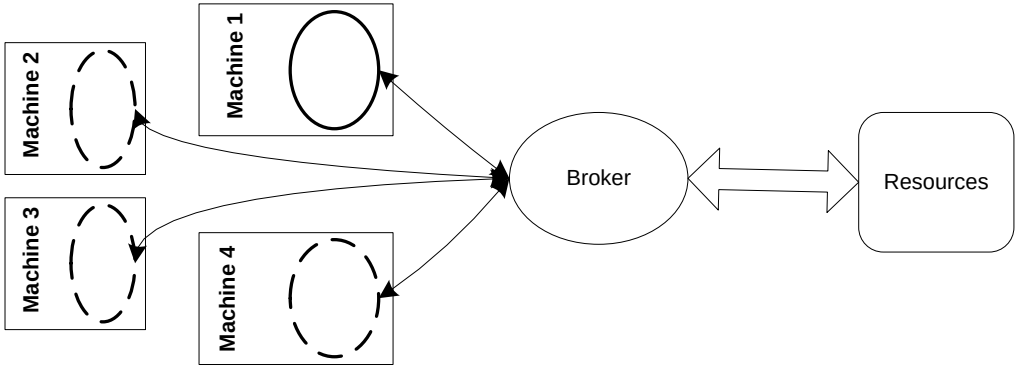
Scalable, Fault-tolerant Management in a Service Oriented Architecture
Harshawardhan Gadgil et. al, Poster HPDC 2007

Performance Evaluation

Measurement Model – Test Setup



Setup A: Running Managers on same machine



Setup B: Running Managers on multiple machine

- Cluster of 8 nodes (Dual Intel Xeon HT CPUs 2.4GHz, 2GB RAM, 1 Gbps, 1.4.2 JVM)
- Multithreaded manager process - Spawns a service specific management thread (A single manager can manage multiple different types of services)
- Limit on maximum services that can be managed
- Limit on maximum number of concurrent requests that can be handled

Performance Evaluation

Results

Scenario illustrating a case with multiple concurrent events

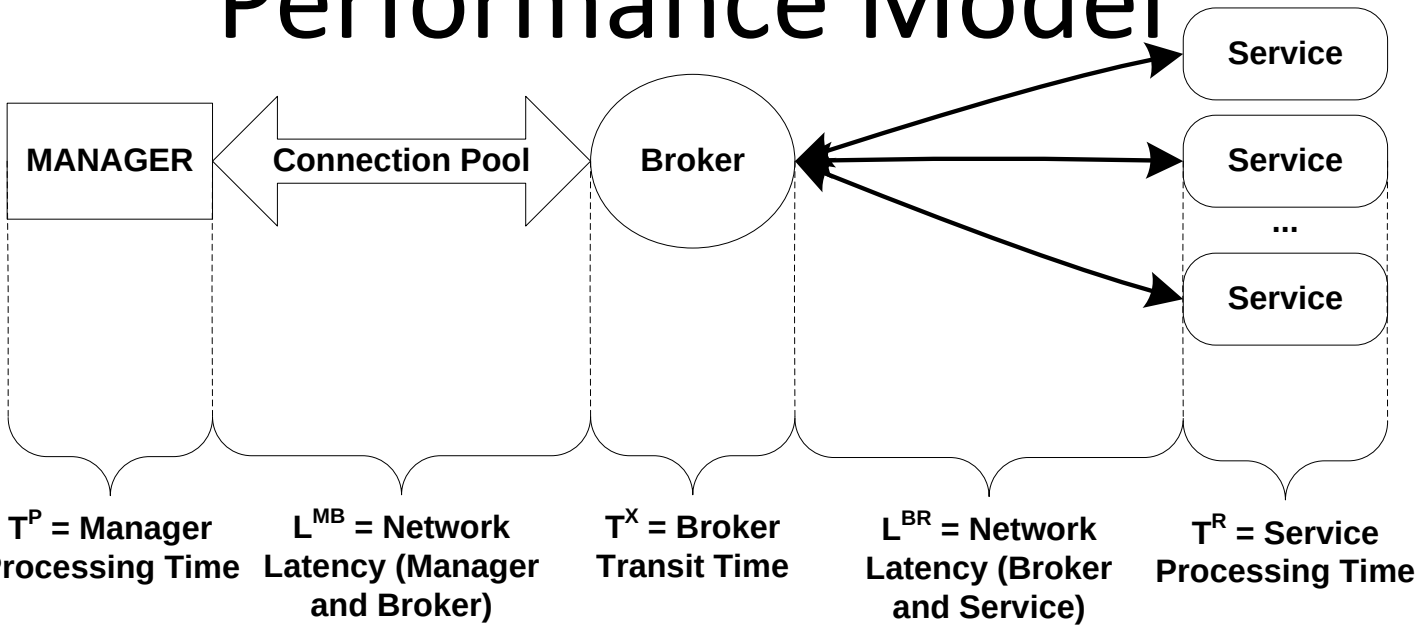
Response time increases with increasing number of concurrent requests

Response time is SERVICE DEPENDENT and the shown times are illustrative

Increases rapidly as no. of requests > 210

MAY involve dependency on external services such as Registry access which will increase overall response time but can allow more than (210) concurrent requests to be processed

Performance Model



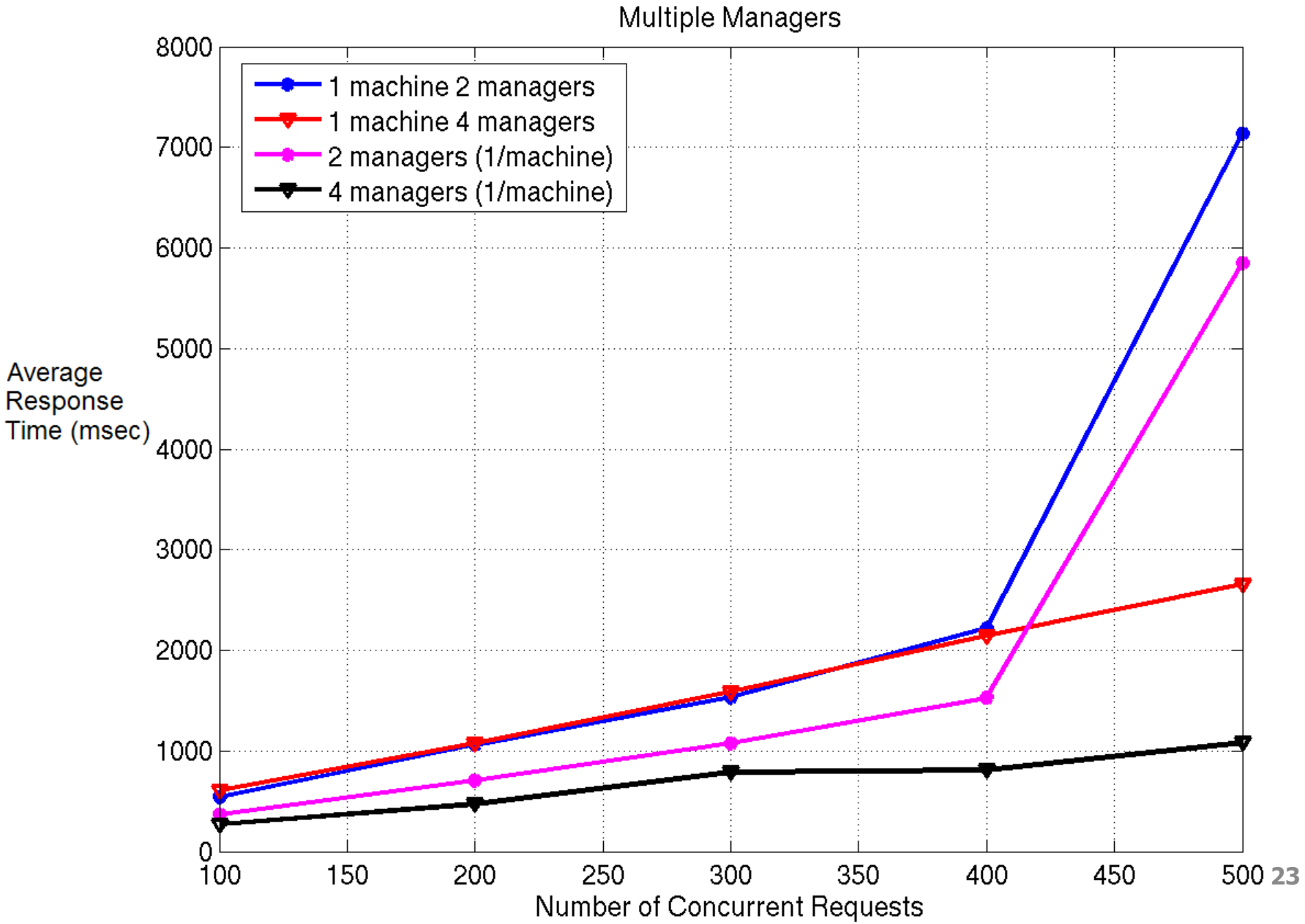
- Avg. Resp. Time = $T^P + [T^R + 2 * (L^{MB} + T^X + L^{BR})]$

  $= T^P + K$

- $T^P = T^{CPU} + T^{External} + T^{Scheduling}$
- $T^{PROC} = (N/C) * T^P$
  - N = Num requests, C = Simultaneous processing
- $D = (C/T^P) * 1000 \approx (2/8.37 \text{ msec}) * 1000 \approx 239 \text{ req/sec}$
  - D = Max requests handled by manager before saturating

Performance Evaluation

Comparing Increasing Managers on same machine w.r.t. different machines



Performance Evaluation

Research Question: How much infrastructure is required to manage N services ?

- N = Number of services to manage
- M = Max. no. of services that connect to a single messaging node
- D = Maximum concurrent requests that can be processed by a single manager process before saturating
  - For analysis, we set this as the number of services assigned per manager
- R = min. no. of registry service components required to provide desired level of fault-tolerance
- Assume every leaf domain has 1 messaging node. Hence we have N/M leaf domains
- Further, No. of managers required per leaf domain is M/D
- Other passive bootstrap nodes are not counted here since << N
- Total Components in lowest level

  = (R registry + 1 Bootstrap Service + 1 Messaging Node + M/D Managers)

  * (N/M such leaf domains)

  = (2 + R + M/D) * (N/M)

Performance Evaluation

Research Question: How much infrastructure is required to manage N services ?

- Thus percentage of additional infrastructure is
  = [(2 + R + M/D)*N/M] * 100 / N %
  = [(2 +R)/M + 1/D] * 100 %

- A Few Cases
  - If, D = 200, M = 800 and R = 4, then Additional Infrastructure
    = [(2+4)/800 + 1/200] * 100 % **≈ 1.2 %**
  - Shared Registry then there is one registry interface per domain, R = 1, then Additional Infrastructure
    = [(2+1)/800 + 1/200] * 100 % **≈ 0.87 %**
  - If NO messaging node is used (assume D = 200), then Additional Infrastructure
    = [(R registry + 1 bootstrap node + N/D managers)/N] * 100 %
    = [(1+R)/N + 1/D] * 100 %
    ≈ 100/D % (for N >> R)
    ≈ 0.5%

No. of services (N), No. of service assigned to manager (D), Registry Service Instances (R), Max. Entities connected to Messaging Node (M)

Contributions

- Designed and implemented a Service Management Framework
  - Scalable to manage large number of services
  - Tolerant to failures in framework itself
    - Can handle failures in managed services via user defined policies
- We have shown that Management framework can be built on top of a publish subscribe framework to provide transport independent messaging between framework components
- Implemented Web Service Management to manage services
- Detailed evaluation of the system components to show that the proposed architecture has acceptable costs

Future Work

- Apply the framework to broader domains

- Investigate application of architecture where significant runtime state needs to be maintained
  - Higher frequency and size of messages
  - XML processing overhead becomes significant

- Investigate strategies to distribute framework components (load balance) considering factors such as locality of resources and runtime metrics

Thanks

Questions / Comments ?