A Scalable and Robust Coordination Architecture for Distributed Management

Srinath Perera, Dennis Gannon
Computer Science Department
Indiana University
Bloomington IN 47405
{hperera, gannon}@cs.indiana.edu

Abstract

While opening avenues for unlimited possibilities, distributed systems have introduced management complexity as an unfavorable trait. Therefore, as distributed systems become commonplace, the automation of system management has become a primary challenge in information technology. The state of art in system management assigns each managed resource to an external entity (manager), which monitors, analyzes and controls the resource and a collection of such managers manages a system. In such settings, each manager has to act with partial knowledge about the system, and to maintain the system as a whole in acceptable state, those managers should be controlled and coordinated. This paper presents a scalable and robust coordination architecture for distributed management. The proposed architecture consists of a cloud of managers placed on a P2P network, and a coordinator, which re-elects on failure. Each resource in the system is assigned to a manager, and managers monitor the system and maintain a distributed data model, which reflects system state (a meta-model). Using the meta-model, each manager enforces a set of user-defined management rules to implement resource level management, and the global coordination is achieved using user-defined, global management rules enforced by the coordinator. Main contributions of the paper are, a coordination architecture for distributed management which supports elections based recovery, a meta-model which reflects the system state, and the application of rules on top of the meta-model to achieve manager coordination.

1. Introduction

Traditionally distributed computing had chosen to focus on systems that share data (e.g. Internet, Database systems, and Distributed file systems). However, with the advent of computing paradigms like Service Oriented Architectures (SOA) and Grid computing, systems that closely coordinate hundreds to thousands of computers towards common goals are emerging. Distributed workflow systems, computation clouds, and stream processing are few examples of such systems. Furthermore, another notable trend is to replace expensive super computers with groups of small commodity hardware, each costs only minute fraction of the former. For an example, Google has claimed to build their colossal architecture using thousands of cheap commodity hardware.

Unlike their ancestors, those new systems must be closely coordinated to achieve common goals. As a result, failure of a single component would have serious consequences than failure of a single machine in the internet. On the other hand, within a system that has thousands of independent components, failures are norm rather than an exception. For an example, if we built a system with units that has 5 years of mean life, and when a thousand of them are put together, suddenly a unit would start to fail once every two days in average. Given such a gigantic deployment a component may fail due to thousands of reasons, varying from network failures, hard disks, to software errors. To avoid having systems brought down to a grinding halt every couple of days, they need to be monitored and managed.

In such settings, the fate of promising concepts like Grid and Service Oriented Architecture will be decided by the availability of distributed monitoring and management solutions to sustain them. Therefore building reliable and usable monitoring and management systems is a critical prerequisite for future distributed systems.

1.1. Distributed Management

A managed system is comprised of manageable resources. A manageable resource is a part of the managed system that can be remotely managed and reasonably separated as an independent entity. A manageable resource could be defined in different granularities; however, a service represents a typical manageable resource. A manageable resource includes sensors, which monitor the system state, and actuators, which allow a remote authority to con-

trol the resource (e.g. WSDM [5]).

A manageable resource exposes its state as properties, and a remote authority may monitor these properties to approximate the state of the resource. Runtime status (e.g. Up, Overloaded, Down), matrices (e.g. CPU usage, Memory usage, IO traffic) and configurations are examples of resource properties. Furthermore, a resource may expose operations (e.g. shutdown operation) which allow a third party to control the resource. Many management specifications have standardized the interface exposed by a manageable resource. Among them are SNMP [16] for network management, Common Management Information protocol (CMIP) [4] for OSI management and WSDM [5] and WS-Management [6] for Web Services. Furthermore, JMX [34] provides an interface between managed java objects and other protocols like SNMP or WSDM. However, it is worth noting that aforementioned specifications only define the interface between resources and a management system, and details of the management architecture are considered to be out of scope of those specifications (e.g. WSDM [5]).

The management monitors and approximates the global state of a managed system and maintains the global state in acceptable ranges by alerting individual resource states as necessary. This process is presented as sensors, brain, and actuators loop in Autonomic Systems [42], which defines three necessary parts of any management system. Management sensors collect information by inspecting the state of a resource periodically (Pull), or by listening to events generated by a resource. The ability to summarize collected information and derive higher-level health of the system is a primary challenge presented by a distributed management system. It is a responsibility of the decision taking unit (brain) to approximate the current state of the system using the data collected via sensors, and to maintain the system with in acceptable bounds by reacting to changes. The unit selects the best course of actions, and executes those actions using actuators.

1.2. Coordinating Distributed Managers

The management collects information from many nodes, processes them, and performs corrective actions on many nodes. With sensors and actuators in place, each resource is assigned to a manager process, which keeps track of its state and takes corrective actions. In literature, there are many systems that use one centralized manager (e.g. [22, 17, 12, 11, 51, 50, 47]). Even though it would be useful as a proof of concepts, assigning all the resources to a single manager will neither scale for potential systems with thousands of resources, nor avoid single points of failure. Therefore, a scalable management system needs more than a single manager, and management tasks should be distributed among those managers.

However, as pointed out by Hariri et al [24], self managing systems require a global control loop which oversees the system and makes sure different management actions do not get in each others way. When resources are assigned to different managers, each manager would likely have an incomplete view of the system. Therefore they may take conflicting decisions, which leads to a inconsistent system. Consequently, the coordination of different managers to take consistent decisions is a primary challenge pose by the coordination architecture.

Furthermore, a management system itself must be reliable and fault tolerant. As the number of services in a distributed system grows, the probability of a service failure increases. As a result, unless the management architecture is carefully designed, adding management could indeed reduce the system reliability. Therefore, the management architecture itself must be redundant and avoid single points of failure.

In this paper, we present a robust and scalable coordination architecture, which addresses aforementioned challenges. The architecture is based on a management cloud, which comprises of a set of managers and a coordinator. Furthermore it includes a distributed meta-model, which reflects the overall system state, and a rule based decision framework, which provides resource management and manager coordination.

We believe our architecture makes following contributions to theory and practice.

1. A novel coordinator based architecture for coordination of distributed managers, which employs a coordinator election based approach. To the best of our knowledge, we are the first to propose it for a system management framework.

2. A new distributed data model (a meta-model) that reflects the system state, which is built on top of the management cloud.

3. A rule based, distributed, decision framework built on top of the meta-model to achieve manager coordination.

4. A generic and extensible management framework that can be utilized to managed different systems

The next section discusses related works in distributed management, and the section 3 presents the coordination architecture in detail. The following section presents the decision framework, and provides a brief description about rules used in the system. The next section presents initial evaluation results, and finally the section 6 concludes the paper.

	Monitoring Only	One Manager without coordination	One Manager with coordination	Managers without coordination	Managers with coordination
Pull / events	InfoSpect [45], Sophia [54] , Waheed et al [53]	Rainbow [22], Unity [17], RefArchi [12], Autopilot [51]	Naik et al [37]	JADE [13], Tivoli [11]	DMonA [35]
Pub/Sub hierarchy		Scale [50] ACME-CMU [47]		ScaMng [14] DREAM [15] SmartSub [20]	NaradaMng [21]
P2P	PIPER [26]			Automate & Accord [7], Mng4P2P [41] WSRF-Container [43]	
Hierarchy	Monalisa [39] Gangila [18] MDS [57] Paradyn MRNet [57], WRMFDS [55]			HiFi [8], IrisLog [38], JINIFed [9] eXtreme [28] ScWAreaRM [19]	WildCat [27]
Gossip	Astorable [44] GEMS [49]			Galaxy [52]	
Spanning Tree (Network/P2P)				ScaInfoMng [56] ACME [40]	
Group Communication				ReactiveSys [32] Galaxy [52]	
Distributed Queue		AutWFEngine [25]			

Table 1. Categorization of Management systems

2. Related Works

Throughout the decade, there have been numerous efforts to build distributed management systems. However, most of the work provides either a centralized manager (e.g. [22, 17, 12, 51, 50, 47]) or set of managers who independently manage the system (e.g. [13, 11, 14, 15, 20, 7, 41, 36, 38, 8, 9, 28, 19, 52, 33]). Centralized managers suffer from single points of failure; where as decentralized managers who do not coordinate their actions, cannot be trusted with automatic management of a system. The scalable coordination of distributed managers has been a topic, which received less attention, even though existing literature has identified the coordination of managers as an area of paramount importance (e.g. [22]).

In the distributed systems literature, management systems are found under diverse sets of topics. Prominent among them are network management, system management, adaptive systems, and autonomic systems. Among literature surveys exist in those topics, Philippe et al [31] discuss network management systems, and Zanikolas et al [57] provide a detailed discussion on grid monitoring systems. Furthermore, Sadjadi [46] and Ghosh et al [23] discuss adaptive systems and self-healing systems respectively. Furthermore, there have been many efforts to build rule based management systems, and those systems can be used for automation or semi-automation of management of systems. We present a literature survey on such systems in the second part of this section.

2.1. Management Architectures

We have done an extensive literature survey, and the table 1, presents the related work on management systems from a coordination perspective. Rows are organized by different communication patterns (e.g. Publish/Subscribe, Peer to Peer, Group Communication) and columns are organized by functionalities of the system and nature of the coordination it presents. We define a manager as an entity that listens to sensor data and performs actions based on that data.

As shown by the table 1, most efforts either use a centralized manager (column 2) or provide a set of managers who

	Rules	Conflict resolution	Verifier	Batch Mode used?	has meta-model?	Planning
DIOS++ [30]	If/then	yes, use priority	No	Yes, next iteration	Yes	No
Rainbow [22]	If/then		No		Yes	No
InfoSpect [45]	Prolog Like	only monitoring	No	No	Yes	No
Marvel-1995 [29]	pre and post condition		Yes	No	Yes	No
Naik et al [37]		Yes	Yes	Yes	No	Yes
Sophia [54]	Prolog like		No	No		No
RecipeBased [48]	Java Code	Yes		No	No	No
HiFi [8] & DREAM [15]	pub/sub Filters and action	No	No	No	No	No
IrisLog [38]	DB triggers	No	No	No	No	No
ACME [40]	timer, sensor, completion conditions trigger actions	No	No	possible with timer conditions	Yes	No
ReactiveSys [32]	if/then	No	No	No	Yes	No
Policy2Actions [10]	define conditions and actions as a policy	Yes, based on runtime state and history	associated tests decide the activation of actions.		No	Yes

Table 2. Categorization of Rule based Management systems

independently manages the system (column 4). The former approach does not scale well and susceptible to single points of failure, where as the latter lacks the coordination between managers.

In our classification, only three systems are identified to provide distributed coordination. Among them, WildCat [27] is based on an agent framework, which uses a management hierarchy and blackboards at each level of the hierarchy to coordinate with each other. However, it suffers from a single point of failure at top of the hierarchy. Top-level managers control next levels by modifying policies; however as authors pointed out, the scalability of blackboards is not clearly established.

DMonA [35] provides a fully decentralized coordination model for network management. However as autonomic systems require a tight global control loop; the fully decentralized approach is unlikely to provide a generic solution for the management coordination.

Gadgil et al [21] provides a replicated management hierarchy, however assumes existence of a centralized, scalable and reliable registry. Furthermore, it uses static management logic tailored for managing messaging nodes and as a result, it is not a generic management framework.

In contrast, the proposed architecture avoids single points of failure using the manager cloud, and the coordination is achieved using global management rules, which are evaluated on top of the meta-model. Furthermore, it is a generic management framework, which can be extended to manage wide range of systems by defining new rules.

2.2. Rule based Management systems

The table 2 presents a summery of rule-based distributed management systems. The first column presents the nature of rules used in the system, and the second asserts the support for conflicts resolution. The third column presents the ability to introduce sanity checks in to the system. Rules are evaluated either using a batch mode, where they are evaluated once per time-period, or whenever an update takes place, and the forth column specifies the choice. If rules are evaluated using a meta-model of the system, rules have a broader view of the system, and consequently provides better coordination among rules. The fifth column asses the existence of a meta-model, and finally, the column six discusses the availability of planning, which would resolves conflicts among actions and selects the best timing to perform them.

In contrast, our architecture uses a Prolog like object oriented rule language (Drools [1]), which performs batch mode rule execution on top of a distributed meta-model of the system. Furthermore, we provide a two level, distributed decision framework, where managers evaluate local rules

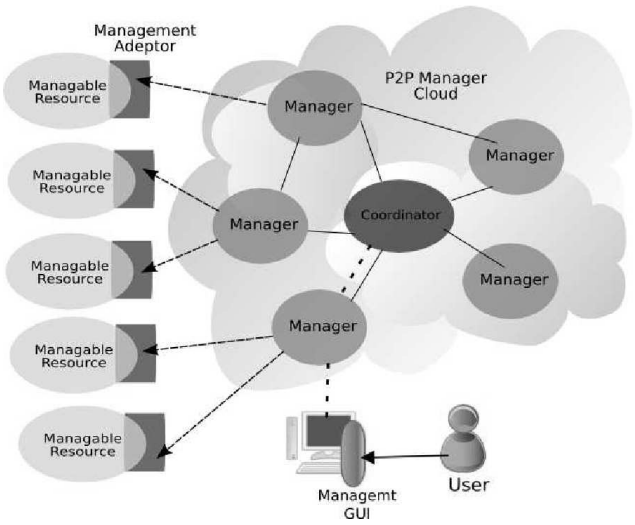

Figure 1. Managment Architecture

and the coordinator evaluates global rules. We expect that, global rules will provide better coordination among components of the system, and due to the distributed nature of the architecture, the decision framework would be more scalable than standalone designs.

3. Coordination Architecture

As shown by the figure 1, the proposed architecture manages a system using a set of processes called managers. There is a special manager called coordinator, which controls and coordinates other managers. The first manager to join the system assumes the role of coordinator, and if the coordinator fails, a new coordinator is elected among live managers. Managers are placed in a P2P network, and that network is used for discovery and election proposes. Each manager sends heartbeat messages to the coordinator, and if a manager fails, the coordinator notices it by the absence of heartbeats, and re-assigns resources that were assigned to the failed manager.

Each managed resource runs a management agent, which advertises the resource at resource start up, and the coordinator assigns the resource to a manager. Each resource sends heartbeat messages to the assigned manager who monitors the resource. A distributed meta-model of the system is built on top of the management cloud, and the meta-model reflects the current state of the system. Furthermore, a rule based, distributed decision framework is built using the management cloud and the meta-model. Listed below are different components of the architecture.

1. A management adaptor tool that can be integrated in to existing services, which enables remote management of those services.
2. A cloud of managers to manage a system, and the cloud consist of a coordinator, which would be re-elected in case of a failure.
3. A distributed meta-model of the system, that represents the current state of the system.
4. A rule based, distributed decision framework that allows users to specify both resource and system level management rules.

3.1. Management Adaptors

The management adaptor is designed as a tool that can be easily integrated with existing services to enable management of those services. The adaptor exposes control and sensor interfaces for the resource using a well-defined Web service management specification, like WSDM or WS-Management, and thereby enables remote management of the given resource. It could be configured or extended to suit needs of each service. Once integrated, the management adaptor intercepts incoming management messages, and redirects them to the underline management logic. Furthermore, it keeps track of state changes of the resource, and sends those changes to the assigned manager, piggy-backed with heartbeat messages. Furthermore, we publish the resource's local management rules as a property of the management resource. As a result, when a resource joins, a manager can inspects the resource and enforces given management rules.

3.2. Management Cloud

The first manager to join the network assumes the role of coordinator, and when a new manager joins the P2P network, it searches the network for a coordinator by performing an anycast. The anycast is done with random waits until a new coordinator is found. Once a coordinator is found, the manager joins the management cloud by contacting the coordinator.

When a resource starts up, it uses a bootstrap P2P node runs on a well-known address to gain entry to the P2P network, and sends a *"Manage Me"* request to a node in the P2P management network. The message is addressed to the coordinator, and any recipient will forward it to the coordinator. If the current coordinator is down, messages are buffered and delivered when a coordinator is elected. When the coordinator receives the request, it assigns the resource to a manager, which in turn contacts the resource. The resource keeps sending *"Manage Me"* requests until a manager is assigned, and the process is restarted when the assigned manager is down.

Elections use the age of managers to decide next coordinator, favoring the oldest manager. Once a manager joins the management cloud, it will be issued a birth-time by the current coordinator, and that birth-time is guaranteed to be higher than all live-managers. Since all birth-times are issued relative to a coordinator, clock synchronization is not expected between managers. The coordinator names his successors (based on birth-time) and propagates them to managers using responses of heartbeat messages. If the coordinator is down, managers detect the failure while trying to send the heartbeat messages, and invite the next successor in line to start an election. If no successor is available, each manager starts an election with random probability to avoid too many parallel elections. Each manager without a known coordinator, reconsiders starting an election once every heartbeat interval. Eventually an election will be started by one of the managers. The election collects birth-times of live managers, and selects the oldest manager. The new coordinator rebuilds the resource assignment by collecting the information from live managers.

In addition, each manager and the coordinator run a daemon, which performs bookkeeping, evaluates rules defined at that level, and carries out corrective actions. The daemon wakes up once per a time-period (e.g. 2 minutes) and this time period is called epoch time.

We have taken a concise effort to make the architecture robust by not making any assumptions about the communication medium reliability or existence of other entities. Most operations of the system are idempotent, and clients retry unsuccessful operations (e.g. search for coordinator, resource sending Manage Me messages, heartbeats, and election).

3.3. The Meta-Model of the System

The management cloud hosts a distributed data model, which is a meta-model that reflects the state of overall system. Each resource of the system is represented by an object in the meta-model, and that object is called a meta-object. Each object contains current values of resource properties exposed by the corresponding resource, and runtime status, number of pending requests, and open connections count, are examples of resource properties. Always, the meta-object of a resource is held by the same manager that manages the resource. The assigned manager listens to heartbeat messages generated by the resource, and keeps the meta-object up to date.

The coordinator provides a uniform interface to the meta-model, which is distributed among managers, and clients contact the coordinator to find the holder of each resource. Each resource is held by it's assigned manager and therefore, the resource assignment table acts as a routing table for the meta-model. In case of a coordinator failure, the elected coordinator will rebuild the resource model by contacting managers in the cloud.

Furthermore, the coordinator maintains a summery of each object locally, and the detailed information about each resource is kept in the assigned manager (the holder). The summery includes few properties like, name, management endpoint and status, thus summery does not take excessive memory, and it is seldom updated. Therefore, the coordinator is expected to hold thousands of resources without excessive overhead. For an example in our implementation, a summery object takes less than 48 bytes, which means one hundred thousand of such objects can be held with just 4.8MB of memory.

The decision framework uses the resource model to evaluate and execute local management rules at each manager, and global management rules at the coordinator. Even though it is a distributed meta-model, most of the time the state of a resource is either accessed by the manager which it is assigned to (which has a local copy), or the coordinator which usually needs summery (which is also available locally). Therefore, we expect remote requests on the meta-model to be rare, and they will not impose scalability concerns. The distributed access is primarily used by GUI based monitoring, however it is possible that rules make distributed references, and those requests will be handled transparently.

4. Decision Framework

As discussed in the former section, the management cloud provides a meta-model of the managed system. The decision framework operates on top of this meta-model, and allows users to express the management logic for the system using a rule language.

The decision framework supports set of actions, and a rule language is used to specify when those actions need to be performed. Among supported actions are, creating a new resource, shutting down a resource, getting or setting the value of a resource parameter, and generating events.

Furthermore, the decision framework supports local and global levels to specify, per resource, and system wide management requirements. Requirements at both levels are specified as management rules, and those rules are evaluated by management demons once per epoch time (e.g. once every 2 minutes), and rules trigger corrective actions. The management daemon at the coordinator evaluates global rules, and daemons at managers evaluate local rules.

Within managers and the coordinator, the meta-model is represented by a in-memory object called meta-model stub. The stub provides access to the system state via in-memory objects, and the decision model uses that state to decide on corrective actions. There is a corresponding in-memory object for each managed resource of the system, and the meta-

model stub acts as a repository of these objects. Management rules are expressed using Drools [1] object oriented rule language, and rules are evaluated on top of the object repository provided by the stub. The program 1 presents

Program 1 A Sample Local Rule

```
rule "ShutDownIde"
    when
        s:Service(idleTime > 1h);
    then
        s.shutdown();
end
```

a sample management rule. This rule searches the meta-model for objects of type *"Service"*, whose *"idleTime"* attribute is greater than one hour, and performs the shut down operation on each matching service.

Resource-level management rules are evaluated by managers. Each manageable resource exposes its own local management rules, and the corresponding manager inspects those rules and manages the resource by evaluating rules once per epoch (e.g. 2 minutes). For an example, the program 1 presents a local rule that shutdown idle resources.

On the other hand, global rules are evaluated in the coordinator, and they are expressed on top of well-defined global parameters of the system. Furthermore, global rules are defined in the system scope, and using the meta-model of the system as the foundation, they provide coordination among different parts of the system. The program 2 presents an

Program 2 A Sample Global Rule

```
rule "CreateAlternativeForRegistry"
    when
        not exists( ManagedService(state == "UpState",
            type == "Registry"));
    then
        system.createService("Registry");
end
```

example global rule. If the system does not have a service of type registry, the rule creates a new registry service. The examples are simplified for clarity and an interesting reader should consult the Drools manual [1] for more information about rules.

5. Evaluation

This section presents initial evaluation results of the management architecture. The evaluation consists of two parts. The first part measures the overhead induce by the management adaptor to a managed service, and the second part observes the ability of management architecture to handle increasing number of resources. The management architecture is implemented using java Web services, and all communications are implemented using SOAP messages sent over HTTP or P2P channels. We have implemented SOAP HTTP communications using XSUL toolkit [3], and P2P communications using FreePastry [2] P2P network.

The first experiment was conducted using a Web service that does 10ms of busy wait on every request. We deployed two copies of the service, one with the management adaptor integrated, and the other unchanged. While varying the number of concurrent clients, throughput and latency, exhibit by each copy was measured. The throughput was measured by each client sending requests for 15 minutes, and measuring the number of successful responses. The latency was measured by each client sending 1000 requests, and calculating the average. Service copies were placed in a 2-processor 3.06GHz Intel Xeon system, running Red Hat Enterprise Linux with 4GB of memory, and requests were sent from four machines.

As shown by the right graph in figure 2, throughputs are almost identical even though the unmanaged service was able to stay slightly ahead. However, the managed service has about 10-20% more latency than the unmanaged version (figure 2, left graph). It is worth noting that the latency exhibits a sub-linear behavior, which is counter intuitive as the latency vs. the number of clients graph must exhibits super-linear or linear behavior. However, as shown by the second Y-axis, about 0-1.5% of all requests failed, and times to receive errors were included in latency. We believe errors account for the sub-linear behavior, and had we ignored failures, that would have affect results as then a client is better off with a timed out request, than a response that takes a long time.

The second experiment presents an initial scalability analysis of the management system. The experiment measures the overhead of monitoring the system and maintaining the meta-model. The experiment was conducted upon a system of services, and the management framework was set up to manage the system. At each test run, each service sent 100 heartbeat messages (one messages per 10ms) to the assigned manager, and the average latency time to process heartbeat messages, and the maximum memory usage of managers was recorded at the end of each test. Measurements were collected while varying the number of services as well as number of managers. The experiment was conducted using computer nodes each having dual AMD 2.0GHz Opteron processor and 4GB Memory. Each manager was given its own node, and 250 services were able to share a single node due to 10ms wait between two heartbeats. Therefore the test with 5000 services took 31 nodes (20 for services, 10 for managers and one for coordinator). Each manager was set up to use 256MB of maximum Heap size, which is the typical java server configuration, and we used Java VM instrumentation to measure the memory consumption.

The figure 3 presents our observations. Some lines stop

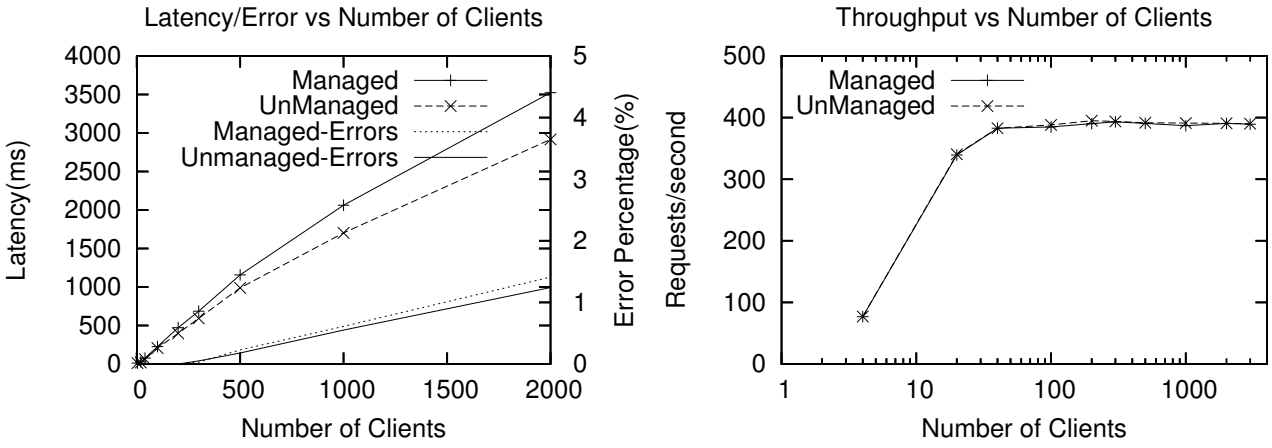

Figure 2. Overhead analysis of Managment Agent

after few readings (e.g. 1 manager line stops after three readings) because the manager ran out of memory. In addition, we have measured the memory consumption of the coordinator, which stayed with in 17-42 MB for all cases. It was evident from results that it is the memory, not the processing time that imposes limits on the architecture, and we are currently exploring possibilities to make our implementation memory efficient. However, we believe current results are promising given that a single manager is able to handle about 500 resources and the system scales up to 5000 resources with 10 managers.

6. Conclusion

The paper presents a robust and scalable coordination architecture for distributed management. The architecture employs a set of managers (manager cloud) to manage a given system, and a coordinator, is used to oversee managers. The coordinator is re-elected in failures, and the new coordinator recovers the manager cloud. Furthermore, a distributed meta-model of the system, which reflects the current system state, is built on top of the manager cloud. The meta-model is distributed among managers, where each manager maintains assigned parts of the model. Moreover, the meta-model provides the foundation for a distributed, rule based decision framework, which consists of two levels. The first level, which resides in every manager, is driven by resource-level management rules. The second level resides in the coordinator, and provides coordination and high level decisions via global rules. The resulting architecture is capable of handling both the resource-level management, as well as the coordination of managers and global decisions. Furthermore, due to the management cloud, the architecture is both robust and scalable. We have implemented and tested the architecture, and experiments have yield promising results, which strengthen our claims.

References

[1] Drools. online. http://labs.jboss.com/drools/.

[2] Freepastry. online. http://freepastry.rice.edu/.

[3] Xml services utility library (version 2). online. http://www.extreme.indiana.edu/xgws/xsul/.

[4] Common management information protocol specification(cmip), 1991. ITU-T Recommendatio n X.711. Data Communication Networks - Open Systems Interconnection (OSI); Management.

[5] Oasis web services distributed management. online, August 2006. www.oasis-open.org/committees/wsdm/.

[6] Web services for management. online, Aprail 2006. http://www.dmtf.org/standards/wsman/.

[7] M. Agarwal, V. Bhat, H. Liu, et al. Automate: Enabling autonomic applications on the grid. In *AMS'03:International Workshop on Active Middleware Services*, page 48. IEEE Computer Society, 2003.

[8] E. Al-Shaer, H. Abdel-Wahab, and K. Maly. Hifi: A new monitoring architecture for distributed systems management. In *ICDCS'99: IEEE International Conference on Distributed Computing Systems*. IEEE Computer Society, 1999.

[9] G. Aschemann, S. Domnitcheva, P. Hasselmeyer, R. Kehr, and A. Zeidler. A framework for the integration of legacy devices into a jini management federation. In *DSOM '99: Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pages 257–268. Springer-Verlag, 1999.

[10] R. M. Bahati, M. A. Bauer, and E. M. Vieira. Mapping policies into autonomic management actions. In *ICAS '06: Proceedings of the International Conference on Autonomic and Autonomous Systems*, page 38. IEEE Computer Society, 2006.

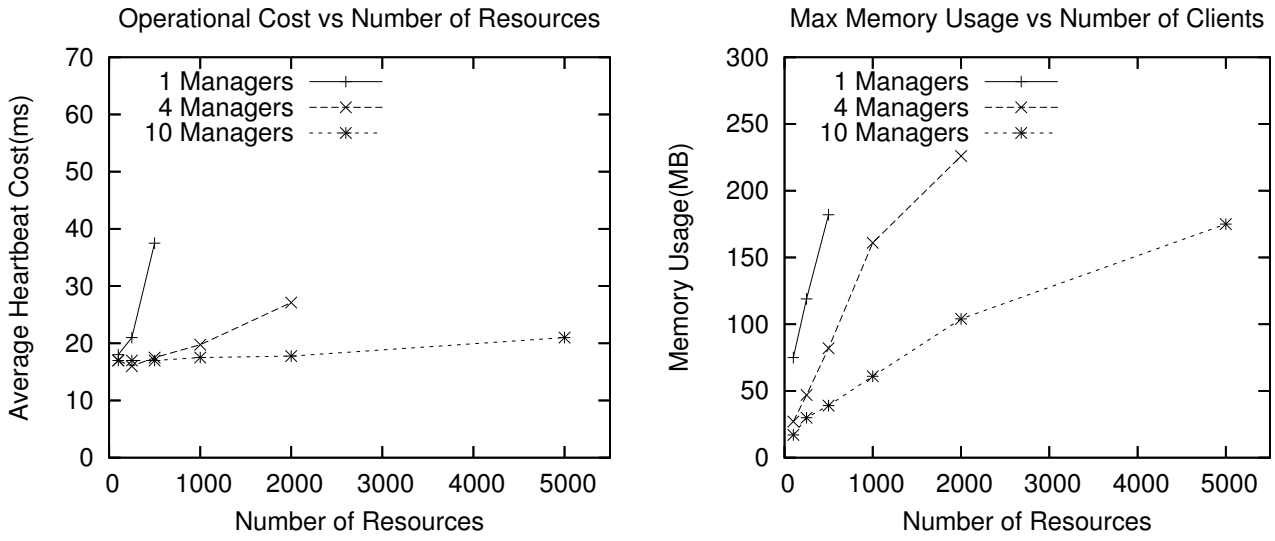

Figure 3. Overhead analysis of Managment Agent

[11] J. B. Baker, D. Reimer, S. Spiro, and J. Whitfield. Management of service-oriented architecture ibm tivoli soa management suite, June 2005.

[12] M. A. Bauer, P. J. Finnigan, J. W. Hong, J. A. Rolia, T. J. Teorey, and G. A. Winters. Reference architecture for distributed systems management. *IBM System. Journal*, 33(3):426–444, 1994.

[13] S. Bouchenak, N. D. Palma, D. Hagimont, and C. Taton. Autonomic management of clustered applications. In *Proceedings of the 2006 IEEE International Conference on Cluster Computing*, pages 1–11. IEEE Computer Society, 2006.

[14] R. P. Brettf, S. Iyer, D. Milojicic, S. Rafaeli, and V. Talwar. Scalable management. In *ICAC '05: IEEE International Conference on Autonomic Computing*, pages 159–170. IEEE Computer Society, 2005.

[15] A. Buchmann, C. Bornhvd, M. Cilia, L. Fiege, F. Grtner, C. Liebig, M. Meixner, and G. Mhl. Dream: Distributed reliable event-based application management.

[16] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. Simple network management protocol (snmp), 1990.

[17] D. M. Chess, A. Segal, I. Whalley, and S. R. White. Unity: Experiences with a prototype autonomic computing system. In *ICAC'04: IEEE International Conference on Autonomic Computing*, pages 140–147. IEEE Computer Society, 2004.

[18] Chun and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[19] M. El-Darieby and D. Krishnamurthy. A scalable wide-area grid resource management framework. In *ICNS'06:International conference on Networking and Services*, page 76. IEEE Computer Society, 2006.

[20] R. E. Filman and D. D. Lee. Managing distributed systems with smart subscriptions. Technical report, 2000.

[21] H. Gadgil, G. Fox, S. Pallickara, and M. Pierce. Scalable, fault-tolerant management of grid services. In *In Proceedings of IEEE Cluster 2007*. IEEE Computer Society, 2007.

[22] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[23] D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya. Self-healing systems - survey and synthesis. *Decis. Support Syst.*, 42(4):2164–2185, 2007.

[24] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu. The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17, 2006.

[25] T. Heinis, C. Pautasso, and G. Alonso. Design and evaluation of an autonomic workflow engine. In *ICAC '05: IEEE International Conference on Autonomic Computing*, pages 27–38. IEEE Computer Society, 2005.

[26] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proceedings of 19th International Conference on Very Large Databases (VLDB)*, 2003.

[27] M. Jarrett and R. Seviora. Constructing an autonomic computing infrastructure using cougaar. In *EASE '06: Proceedings of the Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASE'06)*, pages 119–128. IEEE Computer Society, 2006.

[28] G. Kaiser, J. Parekh, P. Gross, and G. Valetto. Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems. In *AMS'03:International Workshop on Active Middleware Services*, page 22. IEEE Computer Society, 2003.

[29] T. Koch, B. Kramer, and G. Rohde. On a rule based management architecture. In *SDNE '95: Proceedings of the 2nd International Workshop on Services in Distributed and*

Networked Environments, page 68. IEEE Computer Society, 1995.

[30] H. Liu and M. Parashar. Rule-based monitoring and steering of distributed scientific applications. *International Journal of High Performance Computing and Networking (IJHPCN)*, 3(4):78–96, 2005.

[31] J.-P. Martin-Flatin, S. Znaty, and J.-P. Hubaux. A survey of distributed enterprise network andsystems management paradigms. *J. Netw. Syst. Manage.*, 7(1):9–26, 1999.

[32] K. Marzullo and M. D. Wood. Tools for constructing distributed reactive systems. Technical Report TR 91-1193, Ithaca, New York (USA), 1991.

[33] K. Marzullo and M. D. Wood. Tools for constructing distributed reactive systems. Technical Report TR 91-1193, Ithaca, New York (USA), 1991.

[34] E. McManus et al. Java management extensions (jmx) specification. Technical report, 2006.

[35] S. Michiels, N. Janssens, W. Joosen, and P. Verbaeten. Decentralized cooperative management: a bottom-up approach. In *IADIS AC*, pages 401–408, 2005.

[36] P. Murray. A distributed state monitoring service for adaptive application management. In *DSN'05,International Conference on Dependable Systems and Networks*, pages 200–205. IEEE Computer Society, 2005.

[37] V. K. Naik, A. Mohindra, and D. F. Bantz. An architecture for the coordination of system management services. *IBM Syst. J.*, 43(1):78–96, 2004.

[38] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan. Tolerating correlated failures in wide-area monitoring services. Technical report, Intel Corporation, 2004. IRP-TR-04-09.

[39] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. Monalisa : A distributed monitoring service architecture. In *Conference for Computing in High Energy and Nuclear Physics*, 2003.

[40] D. Oppenheimer, V. Vatkovskiy, Hakim, Weatherspoon, et al. Monitoring, analyzing, and controlling internet-scale systems with acme. Technical report, UC Berkelay, 2003. http://techreports.lib.berkeley.edu/accessPages/CSD-03-1276.html.

[41] A. Panisson, D. M. da Rosa, C. Melchiors, L. Z. Granville, M. J. B. Almeida, and L. M. R. Tarouco. Designing the architecture of p2p-based network management systems. In *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 69–75. IEEE Computer Society, 2006.

[42] M. Parashar and S. Hariri. *Autonomic Grid Computing Concepts, Requirements, Infrastructures*. CRC Press, 2006.

[43] C. Reich, M. Banholzer, R. Buyya, and K. Bubendorfer. Engineering an Autonomic Container for WSRF-based Web Services. In *proceedings of the 15th International Conference on Advanced Computing and Communication (ADCOM)*, Bangalore, India, December 2007.

[44] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[45] T. Roscoe, R. Mortier, P. Jardetzky, and S. Hand. Infospect: using a logic language for system health monitoring in distributed systems. In *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 31–37. ACM Press, 2002.

[46] S. M. Sadjadi and P. K. McKinley. A survey of adaptive middleware. Technical Report MSU-CSE-03-35, Computer Science and Engineering, Michigan State University, December 2003.

[47] B. Schmerl and D. Garlan. Exploiting architectural design knowledge to support self-repairing systems. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 241–248. ACM, 2002.

[48] P. Steenkiste and A.-C. Huang. *Recipe based Service Configuration and adaptation*, chapter 10. CRC Press, 2006. Autonomic Computing: Concepts, Infrastructure and Applications, M. Parashar and S. Hariri.

[49] R. Subramaniyan, P. Raman, and A. D. George. Gems: Gossip-enabled monitoring service for scalable heterogeneous distributed systems. *Cluster Computing*, 9(1):101–120, 2006.

[50] W. Vambenepe, C. Thompson, V. Talwar, S. Rafaeli, B. Murray, D. Milojicic, S. Iyer, K. Farkas, and M. Arlitt. Dealing with scale and adaptation of global web services management. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 339–346. IEEE Computer Society, 2005.

[51] J. S. Vetter and D. A. Reed. Real-time performance monitoring, adaptive control, and interactive steering of computational grids. *Int. J. High Perform. Comput. Appl.*, 14(4):357–366, 2000.

[52] W. Vogels and D. Dumitriu. An overview of the galaxy management framework for scalable enterprise cluster computing. *Cluster*, 00:109, 2000.

[53] A. Waheed, W. Smith, J. George, and J. Yan. An infrastructure for monitoring and management in computational grids. *International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, pages 619–628, 2000.

[54] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: an information plane for networked systems. *SIGCOMM Comput. Commun. Rev.*, 34(1):15–20, 2004.

[55] X. Wu, J. Chen, R. Li, and F. LiDOI. Web-based remote monitoring and fault diagnosis system. *The International Journal of Advanced Manufacturing Technology*, 28(1):162–175, 2006.

[56] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 379–390. ACM, 2004.

[57] S. Zanikolas and R. Sakellariou. A taxonomy of grid monitoring systems. *Future Gener. Comput. Syst.*, 21(1):163–188, 2005.