STRUCTURED NAMING AND PEER-TO-PEER DISCOVERY OF RESOURCES IN GRID APPLICATIONS

BY

ADEEP SINGH CHEEMA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Masters of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

Grid computing is a revolutionary new technology that has sparked significant interest in research and scientific communities in the past decade. The Grid is a vast service for sharing resources, such as computational power and data storage, over the Internet. A critical aspect of Grid computing is the collection and indexing of these resources thereby making them accessible to a growing user community comprising of physicists, astronomers, bio-informaticians etc.

The major contribution of this work is the design and analysis of a novel resource discovery protocol intended for Grid applications. Based on the fundamentals of peer-to-peer computing, this resource discovery scheme uses well-established ideas from one domain of distributed computing to surmount shortcomings native to Grid computing. The resulting solutions are elegant and facilitate both multiple attribute and continuous value queries. Trace-driven experiments presented demonstrate that the proposed solutions are both efficient and practical.

To my Parents and Sister

Acknowledgements

First and foremost I would like to thank my advisor Professor Indranil Gupta for his invaluable guidance and for providing the foundations for my research. I would like to thank Muhammad Moosa for all his inputs and suggestions. I am very grateful to Dana Kennedy and the entire staff at the CS Academic Office for their assistance. My family has been a source of constant love, support, guidance and great advice. I would also like to thank all my friends during these four years at the University of Illinois, for making college a pleasant journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Distributed Computational Grids and Peer-to-Peer Systems

Grid computing has come a long way since its inception [1]. One of today's revolutionary fields, the Grid is a vast service for sharing resources such as computational power and data storage for typically high-end applications [2]. Grid computing has an expanding user group that includes physicists, bio-informaticians, astronomers, geologists, the medical community etc. BioGrid, for instance, is a project for developing a Grid specialized in biology and medical science [3].

Grid computing is characterized by its innovative applications and high performance orientation [3-5]. Grid deployments facilitate large scale, flexible, secure and coordinated resource sharing among a dynamic collection of virtual organizations such as individuals, institutions and independent resources. This is in fact the anatomy of the Grid [6].

Peer-to-Peer (p2p) computing is another active research area that shares several computing interests with the Grid [7-11]. However, these communities have remained separate for a while due to different user communities and different research foci. P2p computing, although hampered by a lack of legitimate applications, has acquired a strong footing in certain avenues such as search and storage scalability, decentralization, fault tolerance, anonymity etc. [13]. These avenues are becoming increasingly relevant for Grid computing as it continues to evolve [14-15].

1.2 Problem Statement

The goal of this research is to design a protocol for resource discovery in Grid applications. There are a few well-established characteristics that effective resource discovery protocols for Grid applications should demonstrate. The following characteristics have served as guidelines for this thesis:

- Robustness – Grid applications often desire precision and accuracy. A robust protocol should locate resources when they are present and meet the expectations of the application.
- Scalability – A typical Grid deployment has a vast number of resources and an effective resource discovery protocol must be scalable.

- Efficiency – The protocol should locate resources while consuming a minimal amount of bandwidth. Resources on the Grid have several dynamic properties and a resource discovery scheme should be able to handle this volatility.
- Practicality – Grid applications often specify resources based on multiple criteria and arbitrary scales. These aspects should be addressed in a practical, deployable solution.

1.3 Related Work

This thesis primarily explores two threads of existing work including former research on resource discovery as well on multiple attribute queries.

1.3.1 Grid Resource Discovery

Initial Grid papers [1][3] describe a version of the Grid that uses GRIP, A Grid Resource Information Protocol to register resources on centralized Grid Index Information Servers [16]. Virtual computing toolkits such as the Globus Toolkit also rely on centralized means of storing resource advertisements through their Metacomputing Directory Service, which is layered over LDAP (Lightweight Directory Access Protocol) [17]. Some centralized protocols like Matchmaking, which introduces classified ads, have been successfully deployed on Grid like environments such as Condor [18-21]. Technologies like XML have also been incorporated into service discovery schemes [22].

Several resource discovery schemes [23][24] have also been suggested for specific solutions and environments that tend to rely on unique properties of the domain. There has also been significant work on ontology based resource discovery [23][25] for heterogeneous environments such as pure p2p systems. Finally, ideas on the convergence of the Grid and p2p system relevant to resource discovery have been put forth in recent times [26]. [27] proposes a modified version of Chord for distributed resource indexing on the Grid incorporating some interesting DHT layer optimizations.

1.3.2 Queries on Multiple Attributes

Since DHTs only support efficient single keyword lookups, a naïve approach to resolving a range query is to issue separate point queries to nodes that correspond to each possible value within the query range. This approach becomes quite expensive for a typical sized range query. Efficiently supporting range queries in DHT-based systems has been posed as an open problem in [28-29], with no solution. [30] proposes an adaptive protocol to address this problem in this domain. It utilizes Range Search Trees for content registration and query resolution. Each level of the RST corresponds to a different data partitioning granularity. Registrations are aggregated at different levels of the RST to facilitate queries with different range lengths. Queries are decomposed to a small number of sub-queries for efficient resolution. This scheme is conceptually similar to the Prefix Hash Tree (PHT) mechanism of supporting range queries in [31], with the difference of providing multiple levels to store contents instead of just using the leaf nodes for this purpose.

The authors of [32] investigate the issue of extending P2P-based DHT systems such as CAN to allow range queries by using Space Filling Curves as hash functions [33]. They proposed three simple strategies (Brute Force, Controlled Flooding, and Directed Controlled Flooding) to minimize the communication overhead during the attribute updates. The design of the proposed system is geared towards its application as part of the information infrastructure for computational Grids. Their approach extends some previous solutions such as MDS-2, by adding self-organization, fault-tolerance, and an ability to efficiently handle dynamic attributes, such as server processing capacity. In it, a query is first sent to a node within the range; that node then locally broadcasts the query. Our work is easily adaptable to any type of DHT (based on specific application requirements) and therefore is a more generalized solution. In the sensor networks' domain, [34] proposed a distributed mechanism to partition a multi-dimensional space using a data structure similar to kd-trees to support range queries.

Chapter 2

Modeling a Grid

2.1 Observing Workstations

Prior to designing a resource discovery protocol, it was critical to analyze the nature of these resources in order to represent and model them effectively. This goal was achieved by actively monitoring 6 candidate machines for a period exceeding 3 weeks in an effort to model the availability of individual resources these workstations. This duration amounts to more than 3000 machine hours of activity. These measurements, taken every minute, will be referred to as traces or availability traces through this thesis. The experiments presented in Chapter 4 are driven by these very traces.

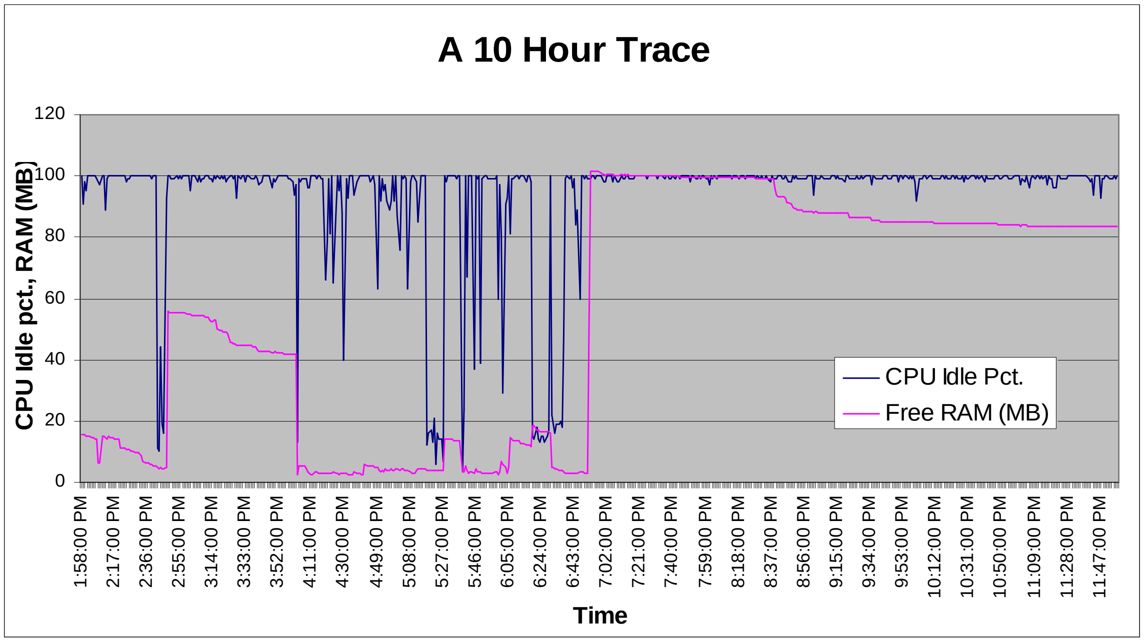


Figure 2.1: A 10 Hour Trace. This plot depicts a sample 10 hours trace taken from an NFS server.

The candidate machines selected included an NFS server, 2 multimedia cluster workstations and 3 other workstations present in the Computer Science domain at the University of Illinois. These machines demonstrate workload characteristics similar to workstations on distributed computational Grids. These characteristics include low average load, relatively short periods of high load interspersed with almost negligible load (typical of batch jobs), temporal patterns as well as maintenance related patterns.

Data collected from these machines, detailed in Appendix A, gives a minute-by-minute representation of CPU usage, RAM availability and disk space availability for each machine. Figure 2.1 is a snippet of this data, representing a 10 hour span on one particular workstation.

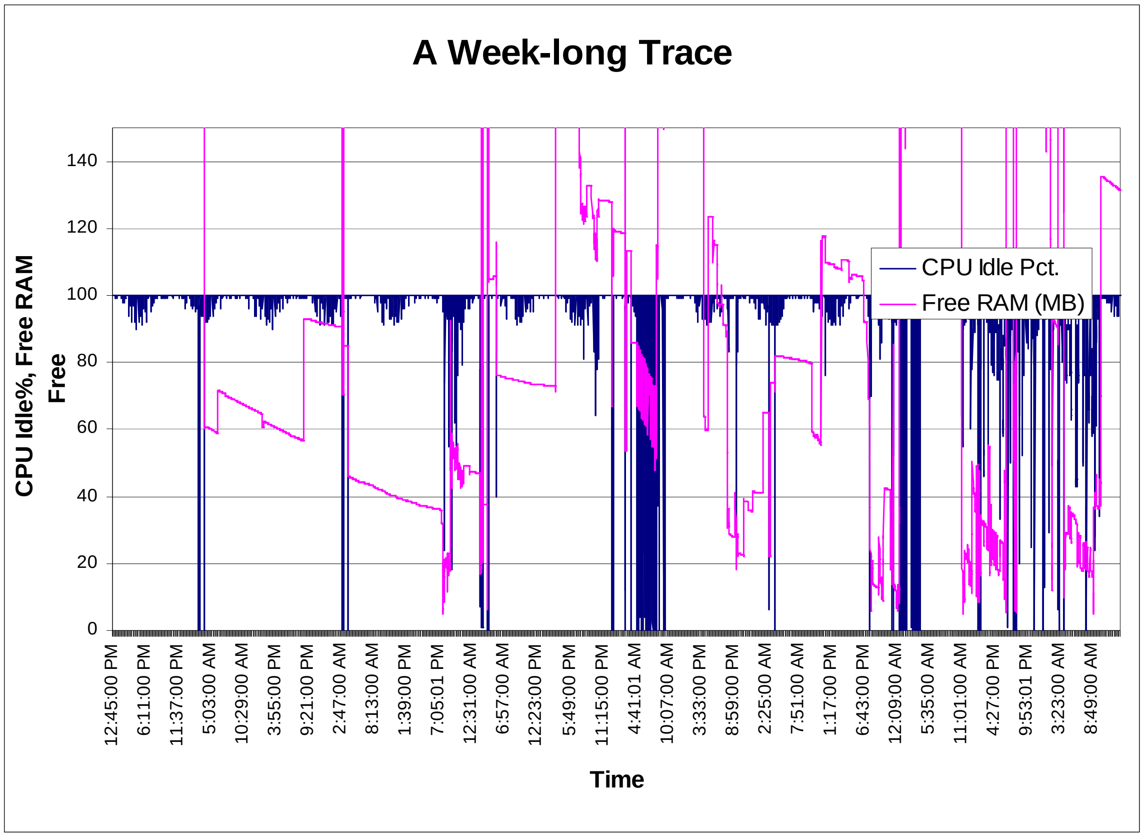


Figure 2.2: A Weeklong trace. This plot is from a weeklong trace of a Linux workstation and depicts several key characteristics including (bi-diurnal) temporal patterns, maintenance patterns, intermittent high load, low average load etc.

2.2 Analysis and Conclusions

Resource availability traces were helpful in establishing several key facts that became central to the design of the protocol. Firstly, it was evident that in order to best represent a resource on the Grid, we would have to accurately and efficiently store what the machine has to offer in terms of idle CPU cycles, megabytes of RAM and gigabytes of disk space. Out of these parameters, CPU utilization tends to be very dynamic and bursty as can be seen in Figure 2.2. Despite the bursts of high-utilization, the average load on the CPU is less than 4%, which suggests that these spare cycles could be harvested for Grid applications. Disk space on the flip side is fairly stable and does not fluctuate much. The trend for disk space, not plotted, is best described as linear with a small negative slope, with occasional small positive jumps. Available RAM lies in the middle of the spectrum and varies a reasonable amount with time. We end up giving it less importance than CPU availability however because most off-the-shelf modern workstations have Virtual Memory, which increases the amount of RAM available on demand.

This analysis forms the basis of the resource discovery protocol presented in Chapter 3.

Chapter 3

DHT-Based Resource Discovery

3.1 Distributed Hash Tables

Distributed Hash Tables are systems that allow key based insertion, lookup and deletion of objects in a distributed setting [7-11]. Such peer-to-peer DHTs form the basis of several applications such as file sharing, storage, multiplayer games etc. [35-39]

Pastry [10] is one such DHT that has been adapted for several different applications [36-37]. Each node in Pastry has a unique identifier (Node ID). When presented with a message and a key, a Pastry node efficiently routes the message to the node with a Node ID that is numerically closest to the key, among all currently live Pastry nodes, which can be visualized as a ring. Each Pastry node keeps track of its immediate neighbors in the Node ID space, and notifies applications of new node arrivals, node failures and recoveries. Pastry takes into account network locality and is completely decentralized, scalable, and self-organizing; it automatically adapts to the arrival, departure and failure of nodes. The expected number of routing steps is O(log N), where N is the number of Pastry nodes in the network.

3.2 The Naming Problem

Any workstation on a distributed computational Grid can be summarized using two sets of attributes shown in Table 3.1.

Resource Attributes	
STATIC Part	**DYNAMIC Part**
OS Configuration	CPU Idle %
CPU Speed	Available RAM
RAM	Available Disk Space
(Max.) Disk Space	

Table 3.1: Static and dynamic attributes of workstations.

The naming problem consists of mapping resources to points on a DHT given that each resource is represented by a collection of attributes. Moreover, the static part attributes of a resource description are discrete and infinite while dynamic attributes are continuous and thus inherently lie on an infinite scale.

The tactic adopted by the suggested protocol involves combining the static and dynamic part of a resource’s attributes into a Resource ID, which serves as a key for DHT operations. Moreover, this encoding should allow for the efficient and practical discovery of these mapped resources.

3.3 Representing Resources

Both the static and dynamic parts of resource attributes can be translated into a format more suitable for indexing. This does come at the cost of reduced precision although the specific trade-off is variable and can be adapted depending on requirements.

STATIC Part of Resource Attributes				
Attribute	**# Bits**	**Range**	**Granularity**	**Border Values**
OS_Configuration	8	0-255 numbered configurations	1	N.A.
Max_CPU_Speed	7	0.1-12.6 GHz	100 MHz	0: < 100 MHz 127: > 12.6 GHz
Max_RAM	10	16-16352 MB	16 MB	0: < 16 MB 1023: > 16352 MB
Max_Disk_Space	7	10-1260 GB	10 GB	0: < 10 GB 127: > 1.26 Tb

Table 3.2: Encoding of static workstation attributes.

DYNAMIC Part of Resource Attributes		
Attribute	**# Bits**	**Quantum**
CPU_Idle_Pct	16	0.006 %
RAM_Free_Pct	10	0.097 %
Disc_Free_Pct	6	1.58 %

Table 3.3: Encoding of dynamic workstation attributes.

Tables 3.2 and 3.3 depict sample representations for both sets of resource attributes for our proposed system. The suggested encoding of the static part of resource attributes represents the set of workstation configurations most likely to be encountered in current Grid deployments. Dynamic attributes have the property that they are continuous. They can however be discretized by representing them as a percentage using a specific number of bits per attribute which results in the encoding and precision shown in Table 3.3. This representation is based on the trends seen in workstation traces from Appendix A and analyzed in the Chapter 2. It is safe to encode dynamic characteristics as percentages simply because the percentage value can be combined with the static representation of the workstation to arrive at the exact status of the resource.

3.4 An Abstract Description

The fundamental concept behind the suggested resource discovery protocol lies in exploiting the scalability and efficient indexing capabilities of peer-to-peer distributed hash tables. Given the static attributes of a resource, our goal is to place that information at a point on the ring representing the DHT. Given that a part of this description is dynamic, each different state of availability of a certain machine must be placed on the ring as well. In the proposed system, this is realized by representing resources as potentially overlapping arcs instead of as individual points on the DHT ring. The beginning of each such arc represents a resources static attribute set and the length of the arc signifies the spectrum of dynamic states the very resource can exist in based on the availability of individual attributes.

A key issue here is that the DHT ring contains only a finite number of nodes while there are an infinite number of dynamic attribute configurations. This is however addressed by the encoding provided in Section 3.1. Another notable fact is that the nodes comprising the DHT can have an arbitrary relation with the actual machines on the Grid. Essentially, each machine on the Grid can choose to host anywhere from 0 to a large number of virtual nodes providing complete flexibility.

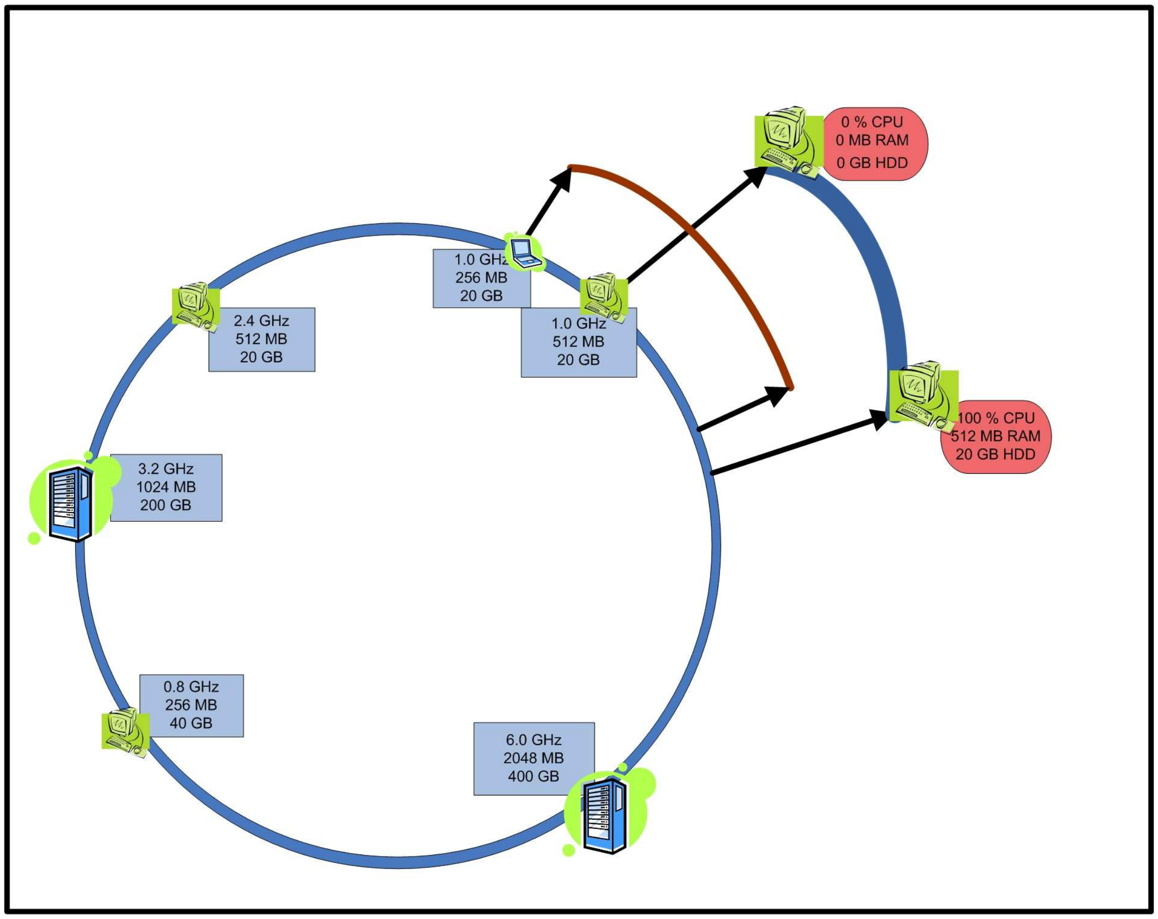


Figure 3.1: Ring Mappings. This is a pictorial depiction of the proposed protocol. The blue ring represents the DHT and neon green Computers represent workstation on the Grid. Blue rectangles (light gray in B&W) are static attributes and red ovals (dark gray in B&W) are dynamic attributes.

This abstract model together with the concrete representation of workstation resources provides the framework for the proposed protocol.

3.5 Initialization and Maintenance

3.5.1 Insertions

Insertion of resources into the suggested discovery protocol has two basic requirements. Firstly, this data should be as uniformly distributed as possible over distributed hash table. This allows for greater scalability. Secondly, each static resource configuration should be represented as an arc representing various degrees of attribute availability.

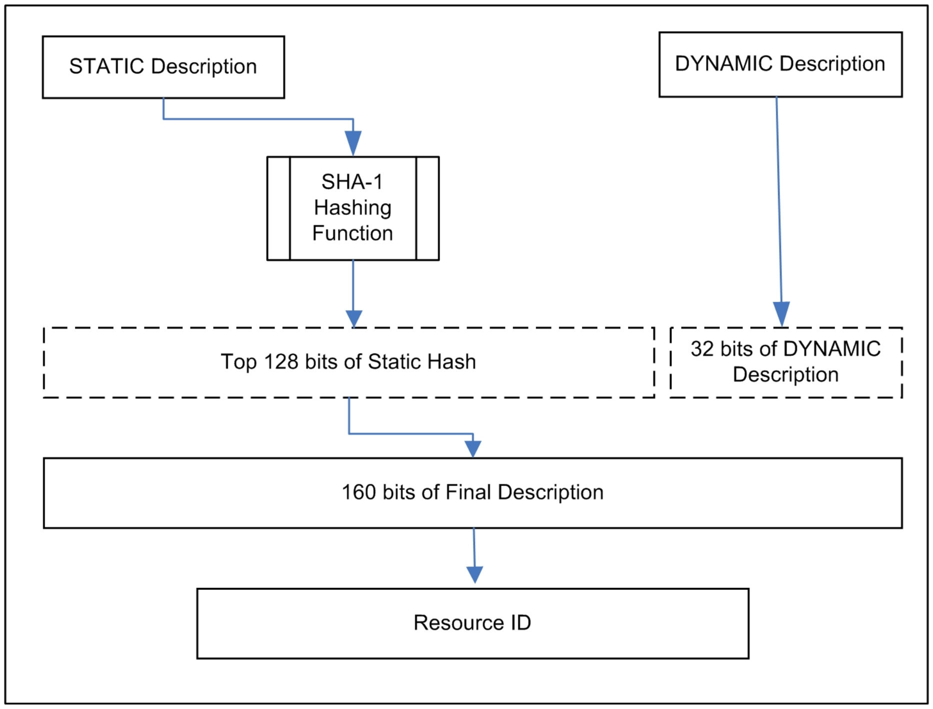
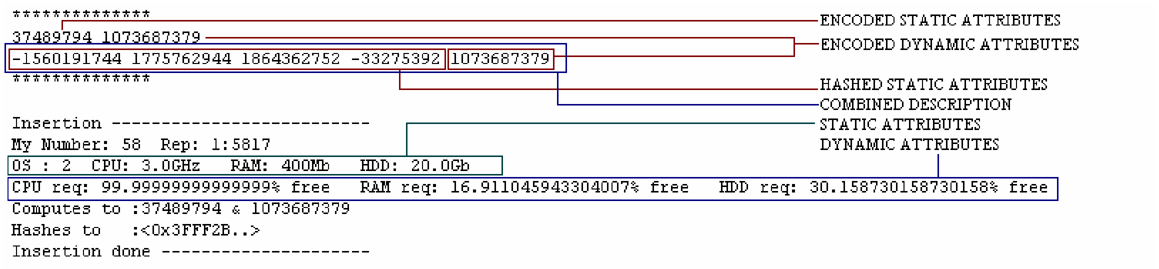

STATIC Description
DYNAMIC Description
SHA-1 Hashing Function
Top 128 bits of Static Hash
32 bits of DYNAMIC Description
160 bits of Final Description
Resource ID

Figure 3.2: Arriving at a Node ID. Attribute descriptions translate into a Resource ID.

These requirements are realized by hashing the 32 bit static part of resource attributes (using a SHA-1 hashing function) resulting in a 160 bit hash. The entire 32 bits of the dynamic part of resource attributes are then appended to the top 128 bits of this static hash. As depicted in Figure 3.2, the result is a 160 bit Resource ID that fulfills both prior criteria and can thus be mapped to a specific node on the p2p distributed hash table. It is possible that the derived Resource ID may not map to a particular node. In that case the underlying DHT routing scheme, such as Pastry described in Section 3.1, automatically routes to the node numerically closest to the calculated Resource ID.



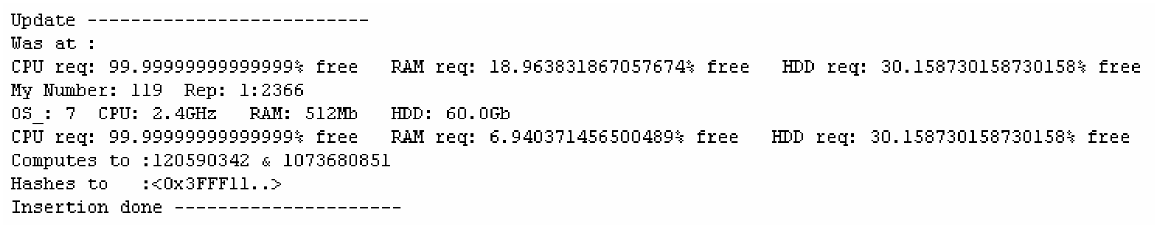
```
***************
37489794 1073687379
-1560191744 1775762944 1864362752 -33275392 1073687379
***************

Insertion --------------------------
My Number: 58  Rep: 1:5817
OS : 2  CPU: 3.0GHz   RAM: 400Mb   HDD: 20.0Gb
CPU req: 99.99999999999999% free   RAM req: 16.911045943304007% free   HDD req: 30.158730158730158% free
Computes to :37489794 & 1073687379
Hashes to   :<0x3FFF2B..>
Insertion done ---------------------
```

Figure 3.3: A Sample Insertion

Insertions are carried out with the help of an INSERT message, which is routed from the node in the p2p network attempting to insert a resource to the node that has a Node ID closest to the calculated Resource ID based on the attributes of resource. Figure 3.3 shows how a resource workstation with its particular static and dynamic attributes maps to a Resource ID given by <0x3FFF2B..> in a 160 bit node namespace.

3.5.2 Updates

Updates are an essential aspect of the protocol given the volatile nature of dynamic resources attributes. Updates are initiated by resources either periodically at rate *URATE* or when they observe a significant change, parameterized in our system as an array *UCHANGE*[], which specifies, as an absolute percentage, how much each dynamic attribute must change for a resource to force an update. For instance, if *UCHANGE[CPU]* is 8.7 then a resource workstation will only issue an update if its CPU's idle percentage varies by at least 8.7%.



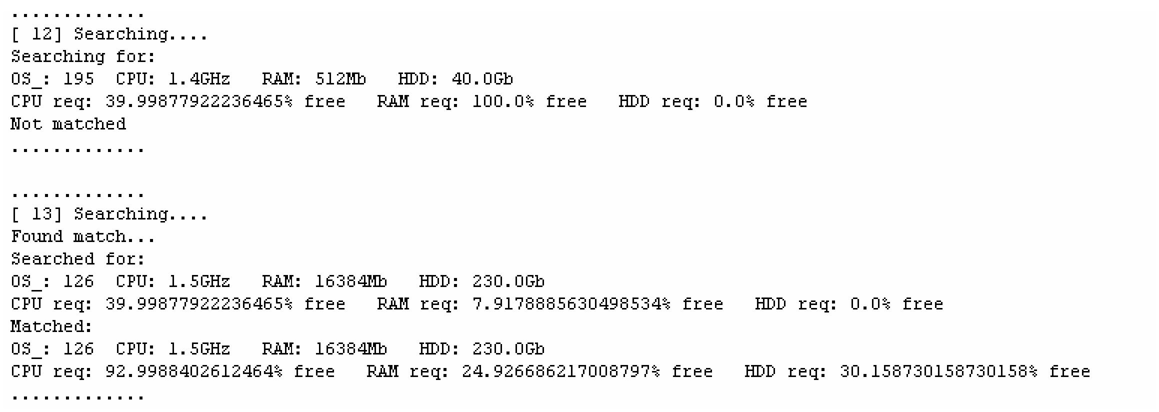
```
Update --------------------------
Was at :
CPU req: 99.99999999999999% free   RAM req: 18.963831867057674% free   HDD req: 30.158730158730158% free
My Number: 119  Rep: 1:2366
OS_: 7  CPU: 2.4GHz   RAM: 512Mb   HDD: 60.0Gb
CPU req: 99.99999999999999% free   RAM req: 6.940371456500489% free   HDD req: 30.158730158730158% free
Computes to :120590342 & 1073680851
Hashes to   :<0x3FFF11..>
Insertion done ---------------------
```

Figure 3.4: A sample update.

Updates are implemented with using an UPDATE message, which is routed from an arbitrary node in the p2p ring to a freshly computed Resource ID. Figure 3.4 shows a sample UPDATE issued by a workstation due to the change in available RAM from 18.96% to 6.94% of its maximum.

It is possible for this update scheme to create stale data in case there is churn in the system or if a resource maps to a different node after an update. This can be avoided by having nodes periodically flush resource entries that have not been updated recently or by sending REMOVE messages to prior node mappings for each UPDATE message.

3.6 Queries



```
.............
[ 12] Searching....
Searching for:
OS_: 195  CPU: 1.4GHz   RAM: 512Mb   HDD: 40.0Gb
CPU req: 39.99877922236465% free   RAM req: 100.0% free   HDD req: 0.0% free
Not matched
.............

.............
[ 13] Searching....
Found match...
Searched for:
OS_: 126  CPU: 1.5GHz   RAM: 16384Mb   HDD: 230.0Gb
CPU req: 39.99877922236465% free   RAM req: 7.9178885630498534% free   HDD req: 0.0% free
Matched:
OS_: 126  CPU: 1.5GHz   RAM: 16384Mb   HDD: 230.0Gb
CPU req: 92.9988402612464% free   RAM req: 24.926686217008797% free   HDD req: 30.158730158730158% free
.............
```

Figure 3.5: Sample Queries.

Searching with the protocol is message based, and essentially involves locating the node on the DHT that is currently hosting the desired resource, expressed in terms of static and dynamic attributes. There are several search heuristics and a particular one should be picked depending on the requirements of the Grid application. Figure 3.5 depicts two sample (single-shot) queries.

3.6.1 Single-shot Searching

This search heuristic involves calculating the Resource ID of the desired resource based on the protocol's attribute encoding and hashing function. This Resource ID is then used to route to a node (with the closest Node ID), which issues a REPLY if it has information about the desired resource. If the update frequency of dynamic resource attributes is large enough, most single searches will work correctly. A high frequency of updates can also overcome the effects of churn.

Singe-shot searching is desirable when the Grid application implements local strategies for searching. A Single-shot search would then query for a particular kind of resource and report if the search was successful or not giving the calling application the freedom to take any subsequent action.

3.6.2 Recursive Searching

Recursive searching is a TTL (time to live) restricted search that continuously seeks out nodes likely to know about the desired resource by progressively tuning search parameters at each hop. The layout of resources on the DHT together with the format of Resource ID's permits the tuning of the least significant bits of the address, representing the dynamic attributes of the resource. Such tuning can help locate resources, which may not match exactly, but are close approximations of the original requirements.

Recursive searching is slightly more efficient than single-shot searching since it uses a distributed/global search strategy and does not report back to the application at every failed attempt thus saving on return messages. Its functionality can however be simulated with multiple single-shot queries.

3.6.3 Parallel Searching

For most Grid applications, it is acceptable to exceed the requirements requested in a query in case an exact match cannot be found. Parallel searching is a formidable strategy that can be used to hasten such alternative searching if so desired by the application.

Parallel searching simply involves initiating multiple searches in addition to a basic search for the exact requested parameters. The additional searches are spawned with the intention of seeking resources that have better attributes than what is actually requested by the application. Parallel searching cuts down on response time when there are a limited number of resources or high contention.

Chapter 4

Experiments and Results

4.1 Simulation

Experimental results based on software simulations of the suggested resource discovery protocol are presented in this Chapter. The protocol simulation is layered over the FreePastry implementation of the Pastry DHT [10][12].

The trace-based simulation engine is highly parameterized and offers a number of benchmarks. A typical simulation starts by creating a set of nodes, *NUM_NODES*, which comprise the DHT required by the protocol. *NUM_MACHINES* parameterizes the number of resources, represented as computer workstations that are introduced into the system, after being read from an ASCII file. The simulation runs for a period in minutes given by *NUM_MINUTES*. Queries are performed on the system at a rate specified by *QUERYRATE*, which is a number specifying the duration between queries in minutes. Queries arrive at a constant rate unless specified otherwise. Update frequency for resources is parameterized using *URATE*, which is the number of minutes a resource will wait before issuing an update and *UCHANGE*[] which holds the percentage change required for forcing an update for each of the dynamic attributes of the resources i.e., CPU_Idle_Pct, RAM_Free_Pct and Disc_Free_Pct. Both these update parameters have been previously described in Section 3.5.2.

Table 4.1 contains are the default values for simulation parameters, which have been used throughout the following experiments unless specified otherwise.

Number of Nodes	**Number of Resources**	**UCHANGE[] (%)**	**Query Rate (Queries per minute)**
1000	20000	{10, 10, 10}	1

Table 4.1: Simulation parameters

Additionally, the simulation uses an extensive set of workstation resource availability traces introduced in Chapter 2 and provided in Appendix A. These traces determine how the dynamic attributes of resources in the system change. The static parts of resource attributes are generated with realistic assumptions. The distributions for these static attributes are provided in the Table 4.2.

Attribute	# Bits	Distribution
OS_Configuration	8	Uniform over entire range
Max_CPU_Speed	7	Uniform over 0.1-3.6 with 1% outliers.
Max_RAM	10	Uniform over [32, 64, 128, 256, 512, 1024] with 1% outliers
Max_Disk_Space	7	Uniform over 10-400 GB with 1% outliers.

Table 4.2: Static resource attributes.

4.2 Experiments

The following experiments provide an insight into the various characteristics, properties and behaviors of the proposed protocol.

4.2.1 Update Frequency

Updates have a significant impact given the dynamic nature of resource availability. Figure 4.1 shows how the number of updates varies with the *UCHANGE[]* parameter (introduced in Section 3.5.2). The clustering of points drops rapidly as *UCHANGE[]* increases and is linear after the 10% point on the X-axis which suggests that any value that is of *UCHANGE[]* that is 10% or greater will exhibit better performance. This result can be verified from the availability traces provided in Appendix A, which depict how most fluctuations in availability have magnitude less than 10%.

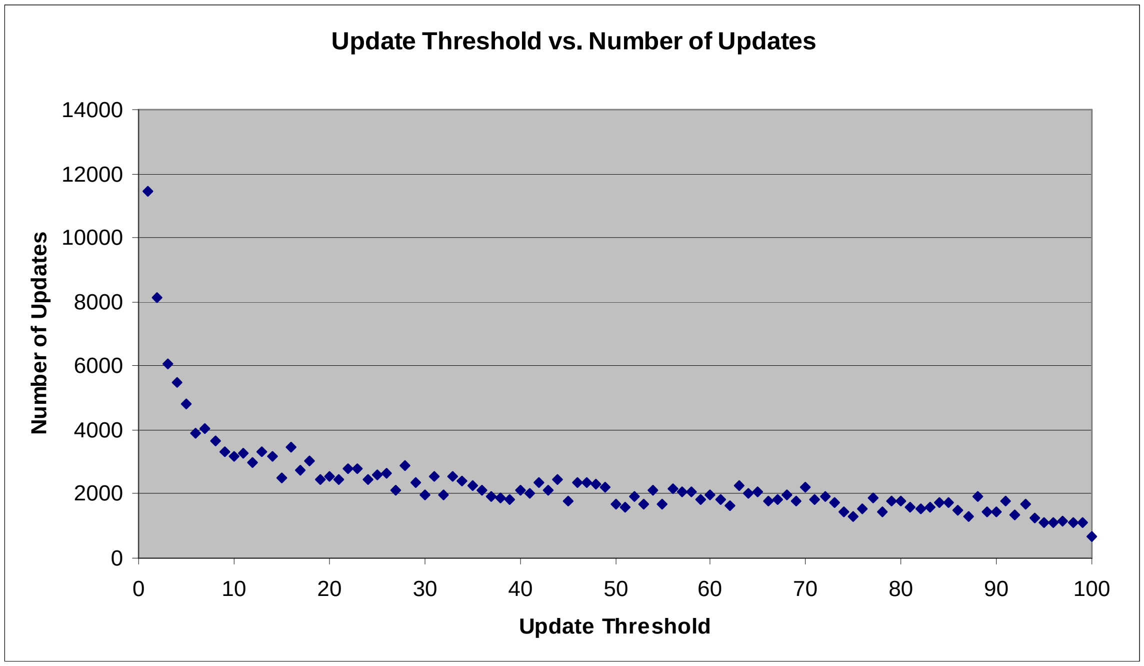


Figure 4.1: Update Threshold vs. Number of Updates. Number of updates is linear after the 10% point when plotted against the update threshold parameter. The 10% update threshold here implies that an update was triggered if any dynamic attributed experienced an absolute 10% change.

Figure 4.2 shows how the rate of updates is constant and independent of the length of the simulation. The large difference in slope between the lines representing 5% and 10% update thresholds ties in with Figure 4.1.

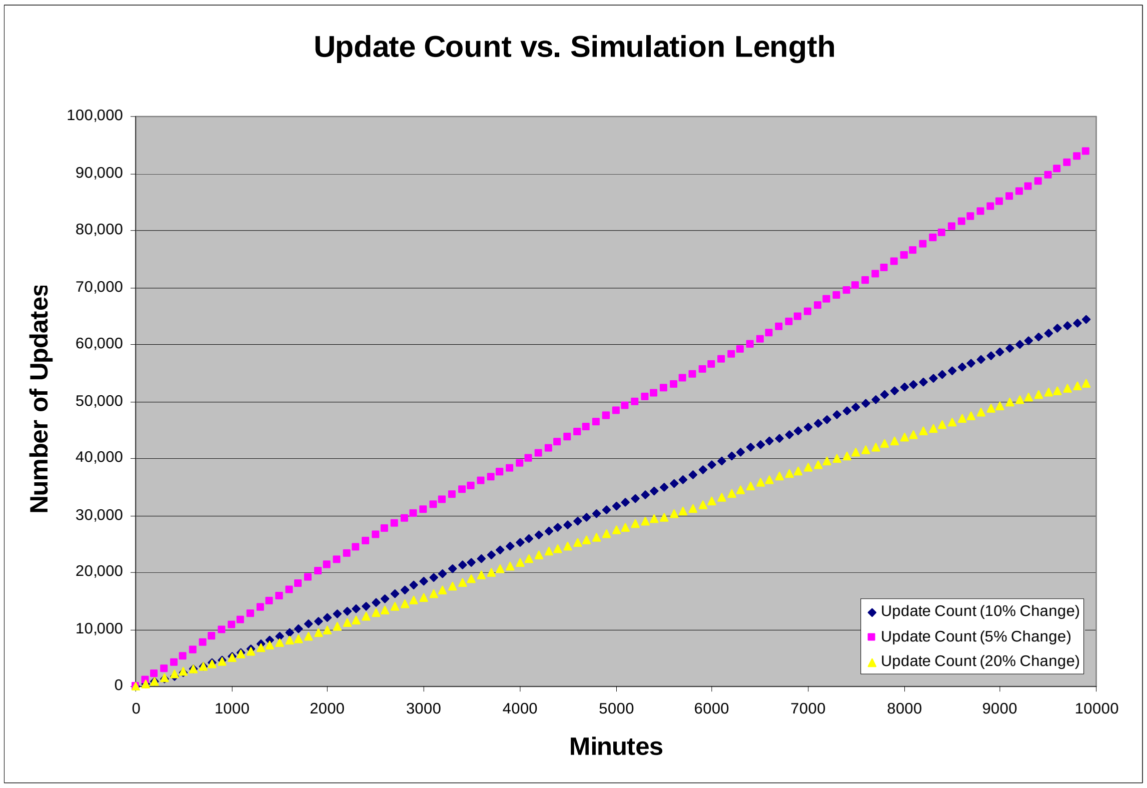


Figure 4.2: Update Count vs. Simulation Length. The rate of updates is constant.

4.2.2 Scalability

Figure 4.3 shows a CDF demonstrating how nodes are hashed based on combined static and dynamic parts of their attributes in the given scheme. This plot is based on static attributes of 20,000 machines where each individual attribute is based on the realistic distributions from Table 4.2. The somewhat asymmetric nature of the CDF can be attributed to these distributions, which drastically bring down the number of possible combinations of static resource attributes. This consequently results in a skewed distribution of nodes. For instance, 30% of the nodes show in the simulation stored 70%

of the data. It is also important to note that that the worst-off node (with the maximum amount of load) has 0.55% absolute load as opposed to an ideal 0.1%.

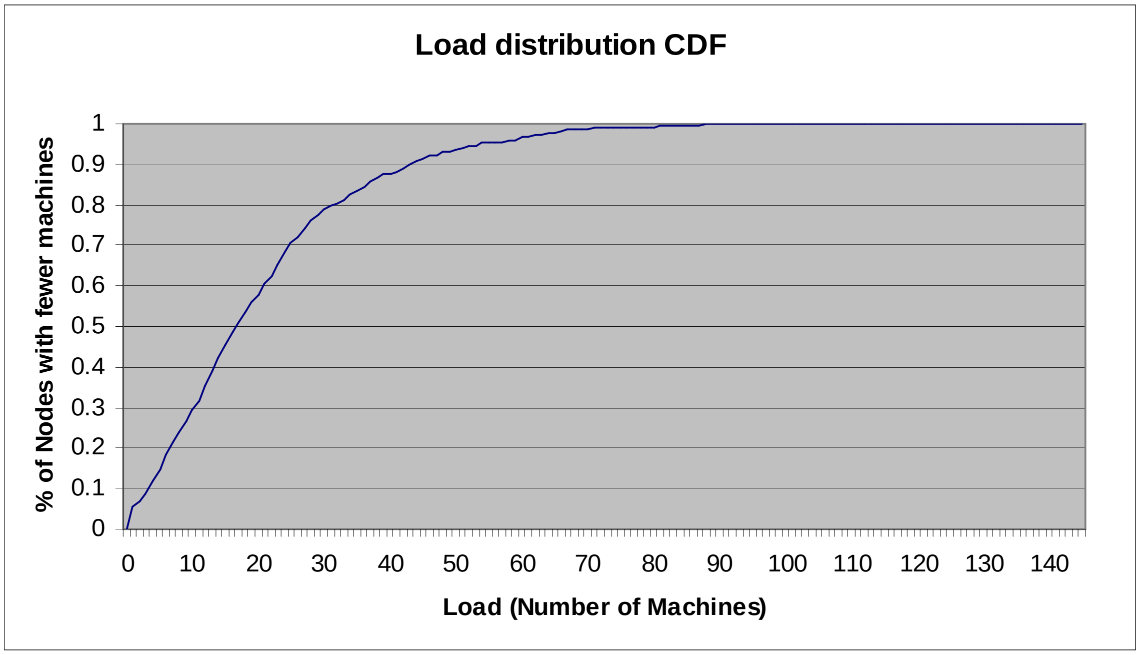


Figure 4.3: Load CDF. CDF of load over DHT nodes

4.2.3 Scalability

Figure 4.4 demonstrates how the bandwidth consumed by the system scales linearly with the number of resources injected into the DHT.

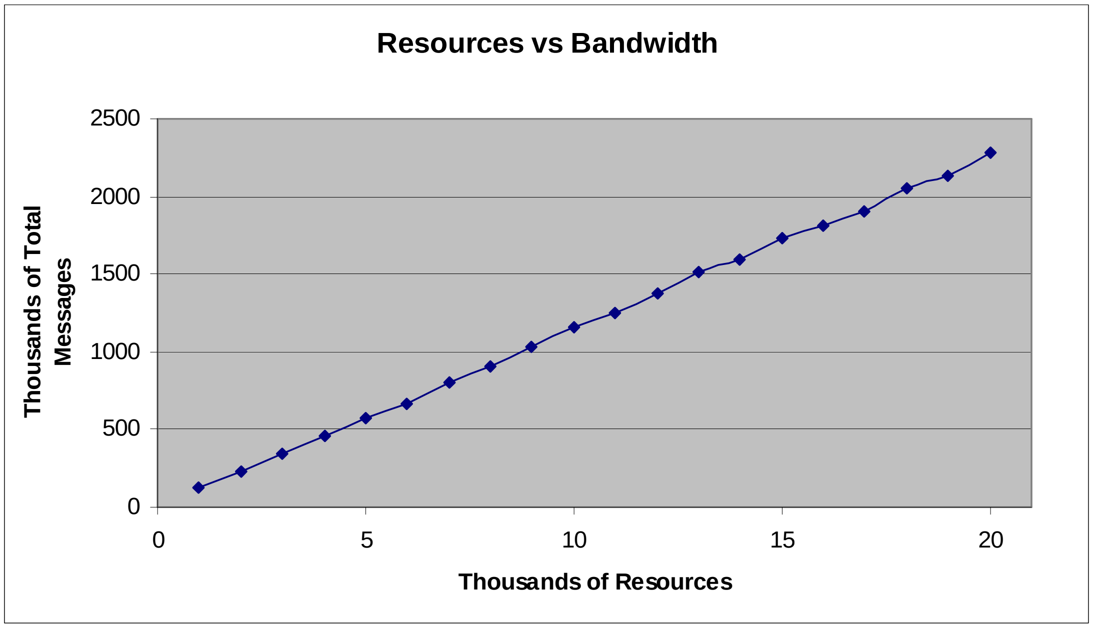


Figure 4.4: Resources vs. Bandwidth. Number of messages increases linearly with number of resources.

4.2.4 Searching

Figure 4.5 shows the linear relationship of search bandwidth with respect to the number of queries while Figure 4.6 shows how the amount of search related bandwidth varies with the inverse of query rate i.e., with the duration between queries. These experiments suggest that the system does not deteriorate with increased rates of querying.

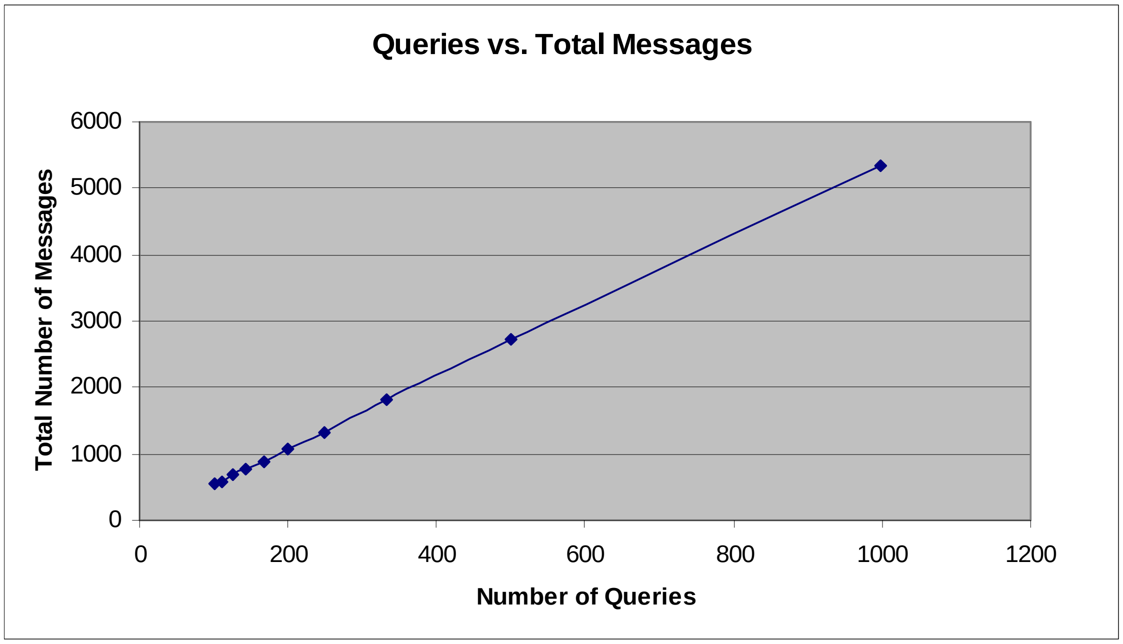


Figure 4.5: Queries vs. Total Search Messages. Number of search messages per query remains constant.

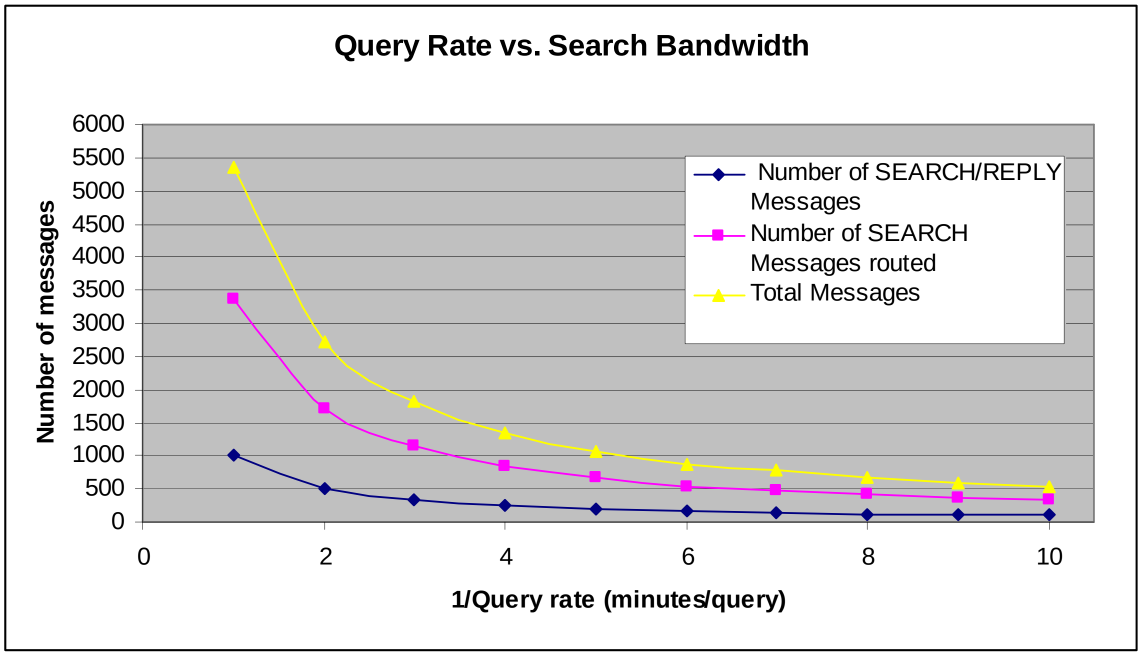


Figure 4.6: Query Rate vs. Search Bandwidth. The number of search messages falls sharply with an increasing query rate

4.2.5 Availability: Short-term Queries

Figure 4.7 shows the relationship between the desired quality of a resource and its availability. Each point on the graph is the result of a single-shot search, which returns the number of machines that match the dynamic CPU attributes from the X-axis for a randomly chosen static description. Due to the skewed distribution of machines over the DHT, these single-shot searches are comprehensive i.e., they return all machines of that static configuration, which also meet the dynamic criteria. The percentage of these nodes is plotted on the Y-axis.

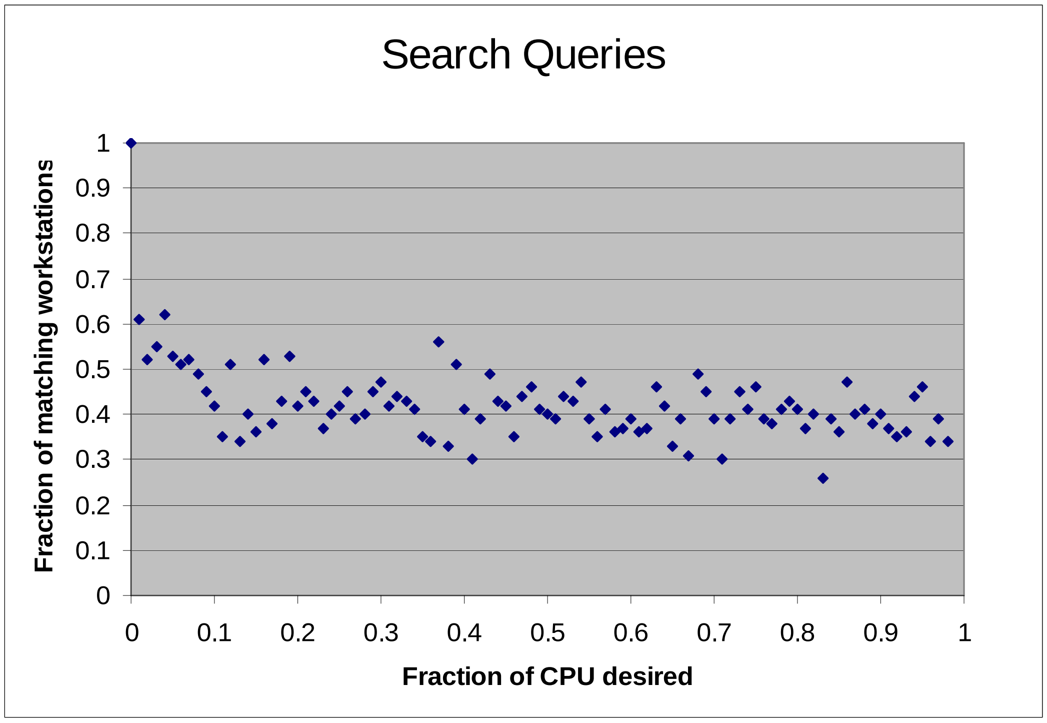


Figure 4.7: Search Queries. Cluster plot representing queries plotted with respect to the number of resources that were fetched together with the quality of resources desired.

Availability: Long-term Queries

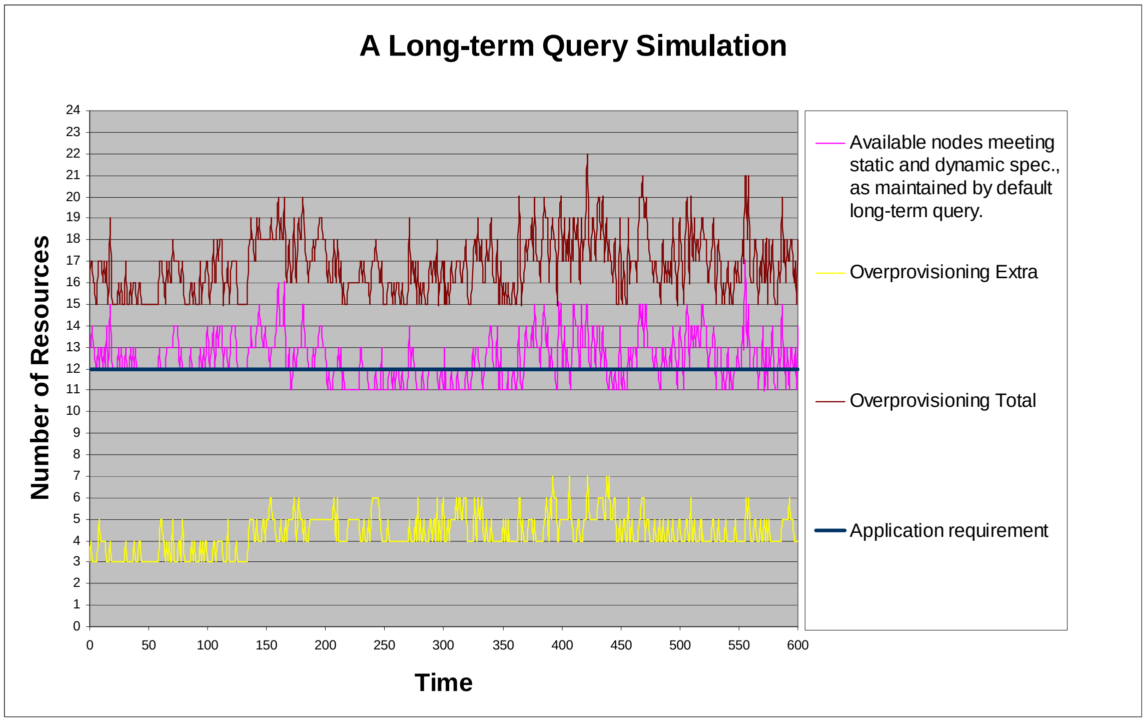


Figure 4.8: A Long-term Query. This ten hour simulation shows how an application requiring a certain number of resources can perform over-provisioning. The application meets its quota by performing parallel searches for resources that exceed its requirements.

Figure 4.8 shows a ten hour simulation of a Grid application using the suggested protocol for resource discovery. The application has a static requirement of 12 nodes of a certain specification shown as the dark straight line on the plot. It tries to meet its requirement by conducting a search on the desired static and dynamic specification. This simulation shows that if that particular resource is limited, then it is possible that the application may not meet its requirement. In this case it is ideal for the application to overprovision i.e., seek out higher quality resources. A combination of true matches and better matches can then be used to satisfy the requirements of the application.

Chapter 5

Conclusion

5.1 Summary

This thesis introduces a protocol for resource discovery in Grid applications, designed on the fundamentals of peer-to-peer computing. The call to unify of these parallel areas has been made in the distributed systems research community and this work advances that cause. The support for queries on multiple attributes and continuous values has been lacking in previous resource discovery and indexing schemes. Both these issues are addressed elegantly in the proposed model. The protocol is put to test with comprehensive and realistic simulations and the results show that the suggested protocol is rigorous, scalable, efficient and practical.

5.2 Future work

This work provides a complete framework for resource discovery in Grid applications. A possible future direction would be to deploy the protocol for an actual application. This will help evaluate its performance based on realistic application requirements and behaviors. The framework for resource discovery itself can be advanced further through more comprehensive integration with its underlying peer-to-peer layer. This unification can make the system more elegant and efficient by eliminating redundancies, for instance, the update messages present in the current protocol can be combined with heartbeating protocols from the DHT layer.

The problem of skewed distribution of resources over nodes can also be addressed by incorporating a load balancing scheme [40]. This will come at the cost of increased bandwidth but may still be desirable under some circumstances.

References

[1] I. Foster, *The Grid: A New Infrastructure for 21st Century Science*, Physics Today, Vol. 55 #2, p. 42, 2002

[2] *CERN GridCafe*. http://gridcafe.web.cern.ch/gridcafe/

[3] The Biogrid Project, http://www.biogrid.jp/

[4] The *GridPhyN* project. http://www.griphyn.org.

[5] *The Human Proteome Folding Project*. http://www.grid.org/projects/hpf.

[6] I. Foster, C. Kesselman and S. Tuecke, *The Anatomy of the Grid Enabling Scalable Virtual Organizations*, International Journal on Supercomputer Applications, 15(3), 2001.

[7] *The Gnutella protocol specification v 0.4*, Document revision 1.2, www.clip2.com.

[8] I. Gupta, K. Birman, P. Linga, A. Demers and R. van Renesse. *Kelips: building an efficient and stable P2P DHT through increased memory and background overhead*. In Proceedings of International Workshop on Peer-to-Peer Systems 2003, pp. 81-86.

[9] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek and H. Balakrishnan, *Chord: a scalable peer-to-peer lookup service for Internet applications*, SIGCOMM 2001, pp.149-160.

[10] A. Rowstron and P. Druschel, *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems*, IFIP/ACM International Conference on Distributed Systems Platforms 2001, pp. 329-350..

[11] *Kazaa*. http://www.kazaa.com.

[12] *FreePastry*. http://freepastry.rice.edu/FreePastry/index.html.

[13] J. Ledlie, J. Shneidman, M. Seltzer, *et al.* et al, Scooped, again, In Proceedings of International Workshop on Peer-to-Peer Systems 2003, pp. 129-138.

[14] M. Roussopoulos et al, 2 P2P or Not 2 P2P, In Proceedings of International Workshop on Peer-to-Peer Systems 2004, pp. 33-43.

[15] I. Foster and A. Iamnitchi. *On death, taxes and the convergence of peer-to-peer and grid computing*, In Proceedings of International Workshop on Peer-to-Peer Systems 2003, pp. 118-128.

[16] K. Czajkowski, S. Fitzgerald, I. Foster,and C. Kesselman ,*Grid Information Services for Distributed Resource Sharing*, IEEE Computer Society, Washington, DC, 2001, pp. 181-194.

[17] I. Foster and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, International Journal on Supercomputer Applications, *11(2):115-128, 1997.*

[18] R. Raman, M. Livny, and M. Solomon, *Matchmaking: Distributed Resource Management for High Throughput Computing*, In Proceedings of IEEE International Symposium on High Performance Distributed Computing 1998, pp. 140-147.

[19] C. Liu, L. Yang, I. Foster and D. Angulo, *A Constraint Language Approach to Matchmaking*, International Symposium on High Performance Distributed Computing (HPDC'02), pp. 63-72, July, 2002

[20] M. J. Litzkow and M. Livny, *Experience with the Condor Distributed Batch System,* IEEE Workshop on Experimental Distributed Systems 1990.

[21] M. J. Litzkow, M. Livny, and M. Mutka, *Condor - A Hunter of Idle Workstations*, In Proceedings of the 8th International Conference on Distributed Computing Systems 1988, pp. 104-111

[22] I. Foster, C. Kesselman, J. Nick and S. Tuecke, *The Physiology of the Grid*, Proceedings of Open Grid Service Infrastructure WG, Global Grid Forum 2002

[23] M. Aktas, M. Pierce and G. Fox, *Designing Ontologies and Distributed Resource Discovery Services for an Earthquake Simulation Grid*, 2004

[24] D. Oppenheimer, J. Albrecht, D. Patterson and A. Vahdat, *Distributed Resource Discovery on PlanetLab with SWORD*, First Workshop on Real, Large Distributed Systems 2004.

[25] F. Heine, M. Hovestadt and O. Kao, Towards *Ontology-Driven P2P Grid Resource Discovery*, Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing 2004, pp. 76-83.

[26] I Foster and A. Iamnichi, *A Peer-to-Peer Approach to Resource Discovery in Grid Environments*, Grid Resource Management, Kluwer Publishing, 2003.

[27] Y. Meng et al, *A DHT-Based Grid Resource Indexing and Discovery Scheme*, Singapore-MIT Alliance Annual Symposium 2005

[28] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. *Complex Queries in DHT-based Peer-to-Peer Networks*. In Proceedings of International Workshop on Peer-to-Peer Systems 2002.

[29] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. *Querying the internet with pier*. In Proceedings of the International Conference on Very Large Data Bases, 2003 pp. 1-11

[30] J. Gao and Peter Steenkiste. *An Adaptive Protocol for Efficient Support of Range Queries in DHT-based Systems*. In Proceedings of The IEEE International Conference on Network Protocols 2004.

[31] S. Ratnasamy, J. Hellerstein, and S. Shenker. *Range Queries over DHTs*. Technical Report IRB-TR-03-009, Intel Corp., 2003.

[32] A. Andrzejak and Z. Xu. *Scalable, Efficient Range Queries for Grid Information Services*. In Proceedings of the IEEE Conference on Peer-to-Peer Computing 2002.

[33] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. *A Scalable Content-Addressable Network*. In Proceedings of SIGCOMM 2001, pages 161–172,

[34] X. Li, Y. Kim, R. Govindan, and W. Hong. *Multidimensional Range Queries in Sensor Networks*. In Proceedings of SenSys'03.

[35] B. Knutsson, H. Lu, W. Xu and B. Hopkins, *Peer-to-Peer Support for Massively Multiplayer Games* The Annual Joint Conference of the IEEE Computer and Communications Societies 2004.

[36] A. Rowstron and P. Druschel, *Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility*, In Proceedings of Symposium on Operating Systems Principles 2001. pp. 199-201.

[37] J. Chase, B. Chun, Y. Fu, S. Schwab, and A. Vahdat. *Sharp: An architecture for secure resource peering*, In Proceedings of Symposium on Operating Systems Principles 2003. pp. 170-184.

[38]J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, *OceanStore: An Architecture for Global-Scale Persistent Storage,* In Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems 2000.

[39]I. Clarke, O. Sandberg, B. Wiley and T. Hong, *Freenet: A Distributed Anonymous Information Storage and Retrieval System,* In Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability

[40] J. Byers, J. Considine and M. Mitzenmacher, *Simple Load Balancing for Distributed Hash Tables.* In Proceedings of International Workshop on Peer-to-Peer Systems 2003, pp. 80-87.

Appendix A

Machine Traces

In order to accurately depict the dynamic characteristics of grid machines, realistic traces of how the individual attributes of workstations vary were collected and analyzed. Over 3000 machine hours of such traces have been accumulated and used in simulations. Traces were collected from 6 machines in the University of Illinois, Computer Science domain. These machines included one NFS server, two machines intended for multimedia use and three workstations. Each machine ran a recent version of the Linux operating system. Traces were collected every minute for a 3 week duration using the Linux *crontab* utility. Each trace captured the machines User and System CPU utilization, available RAM as well as available and utilized disk space using Linux commands *vmstat* for CPU and memory traces and *df* for disk space.

The following Figures A.1-A.6 show CPU and RAM availability for 6 different machines for the first week of monitoring.

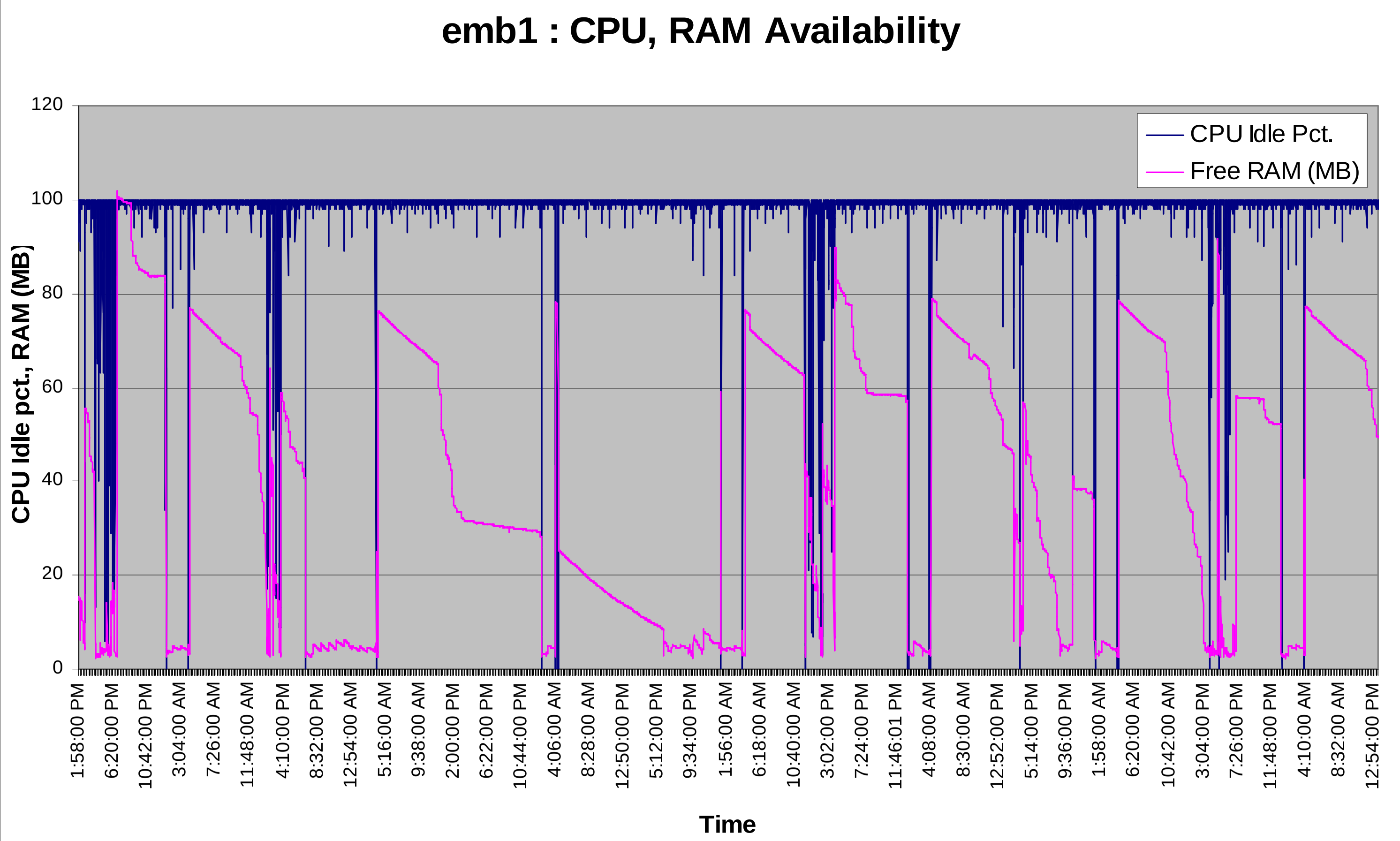


Figure A.1: CPU and Memory traces for emb1

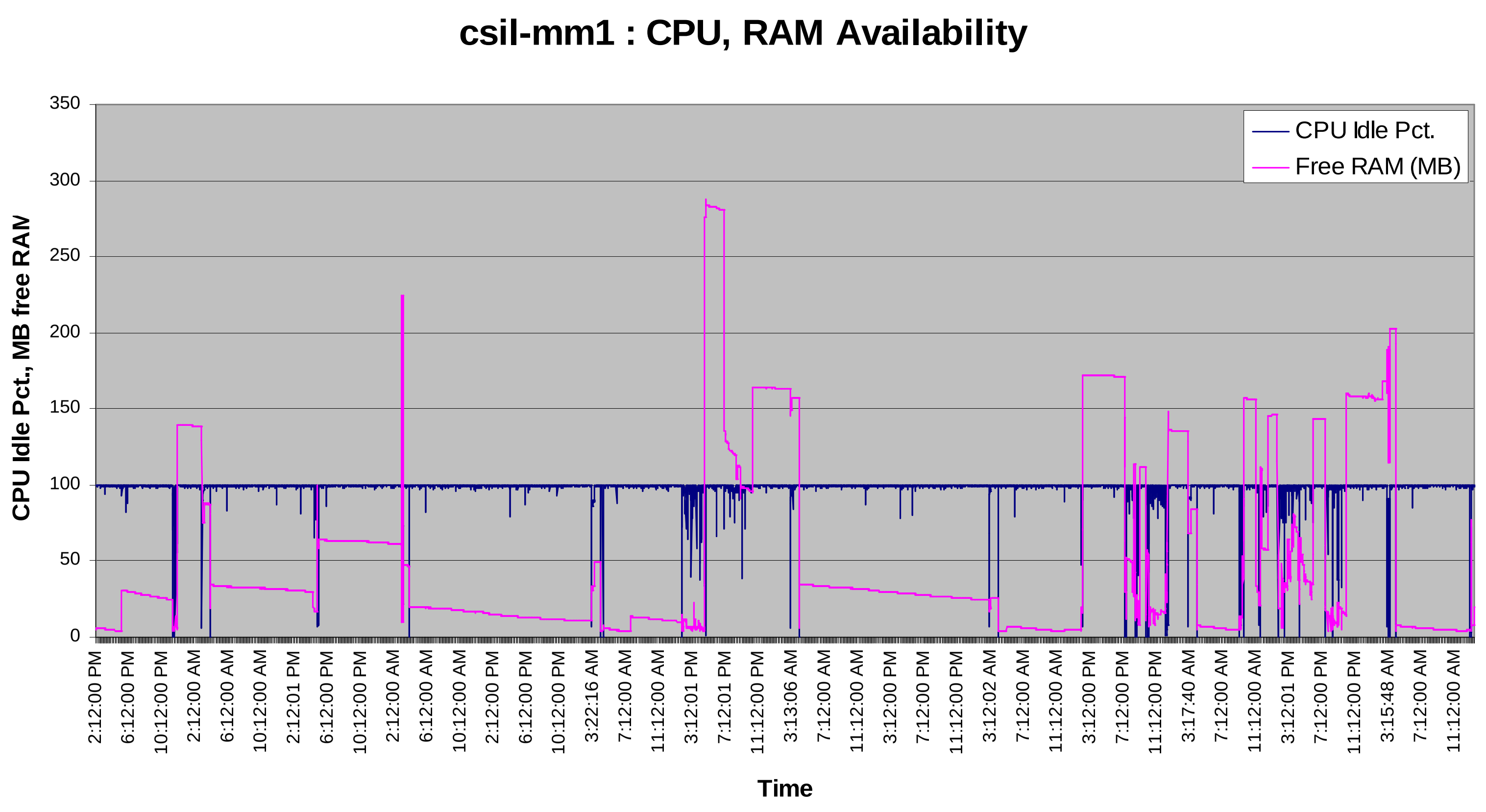


Figure A.2: CPU and Memory traces for csil-mm1

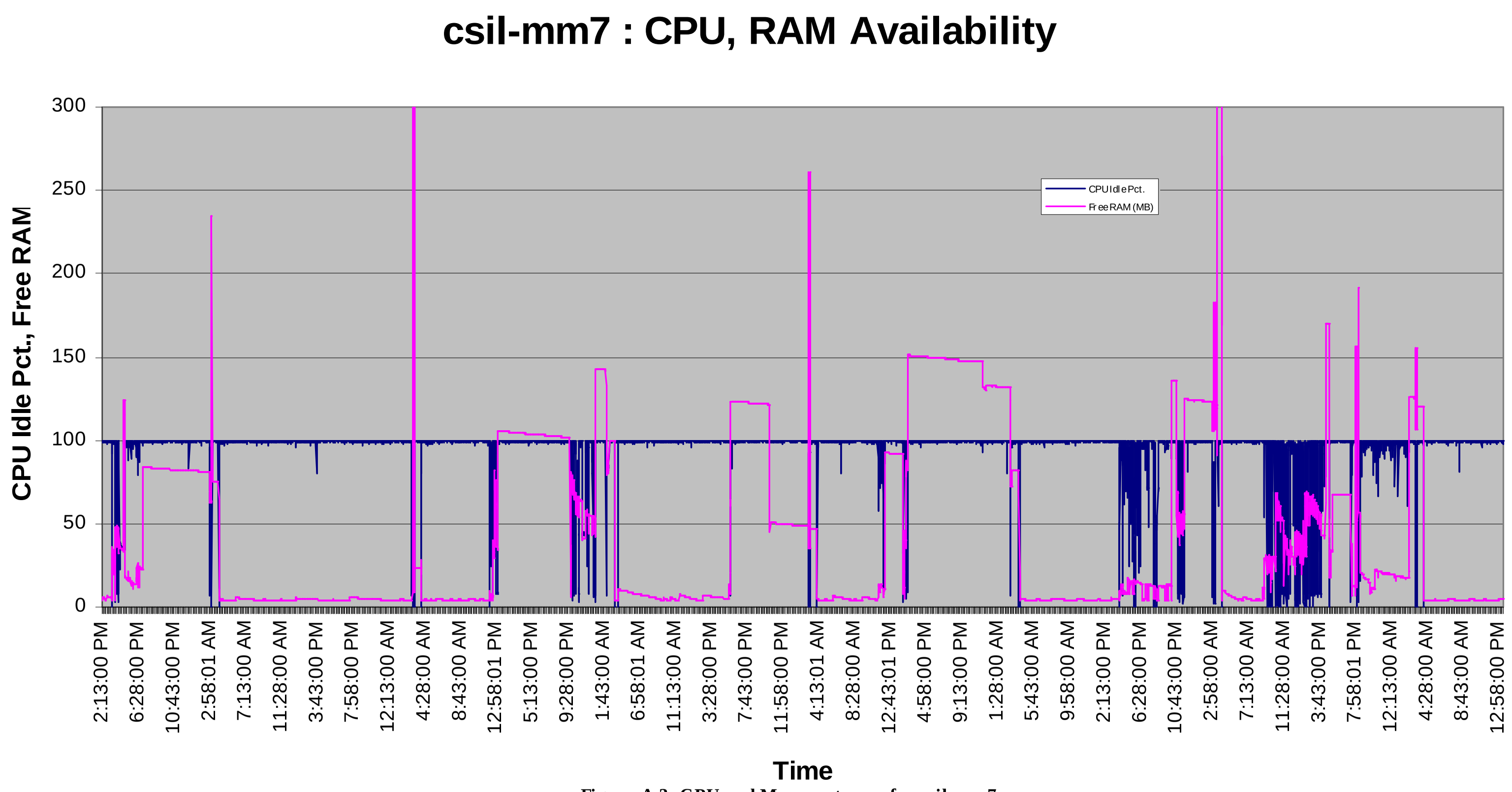


Figure A.3: CPU and Memory traces for csil-mm7

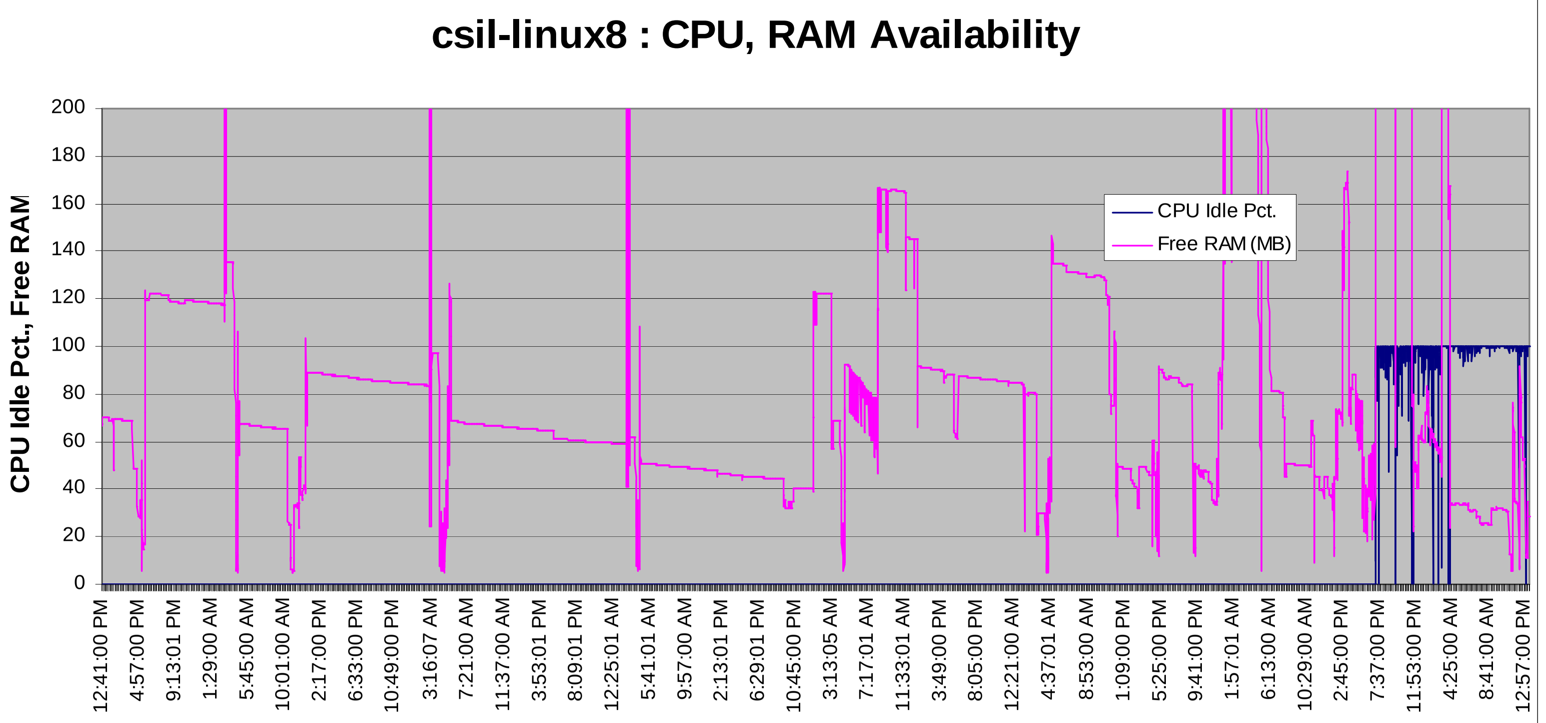


Figure A.4: CPU and Memory traces for csil-linux8. CPU Idle % is 0 for a long time at the start.

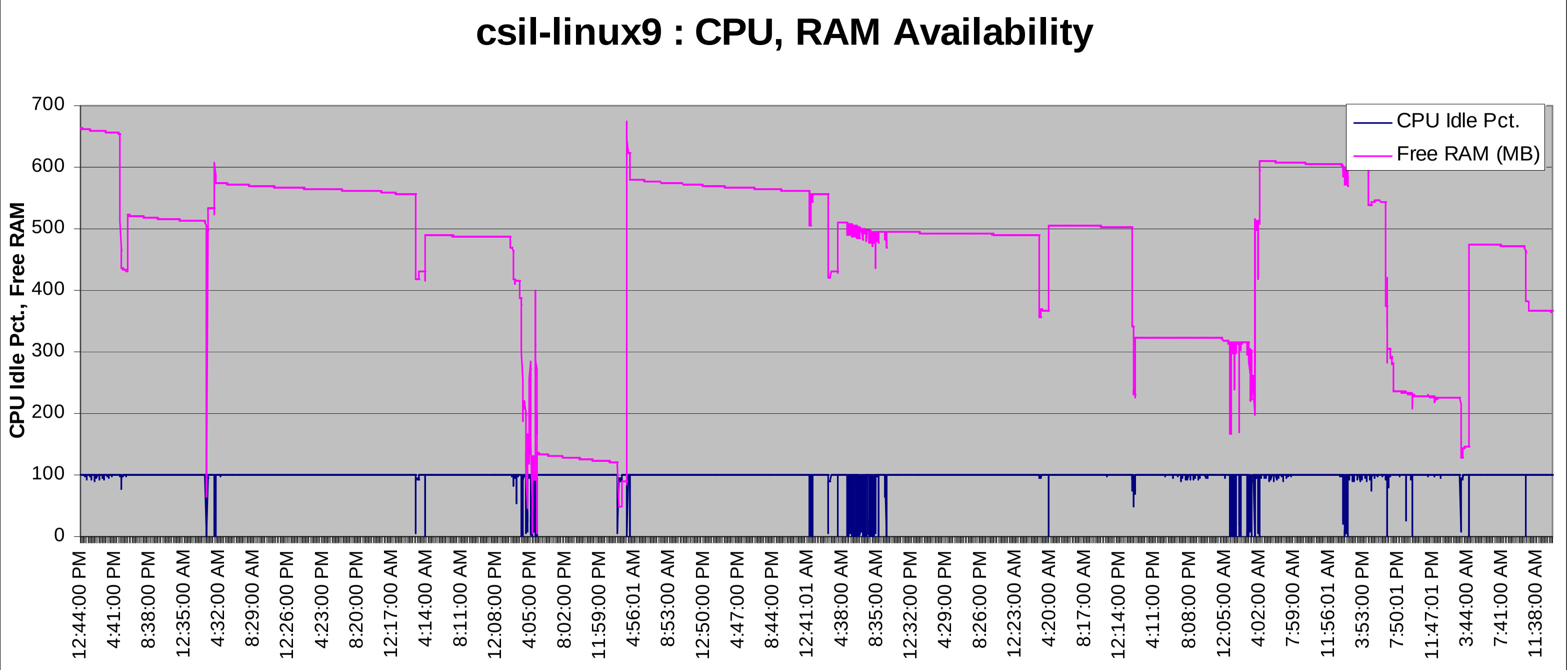


Figure A.5: CPU and Memory traces for csil-linux9.

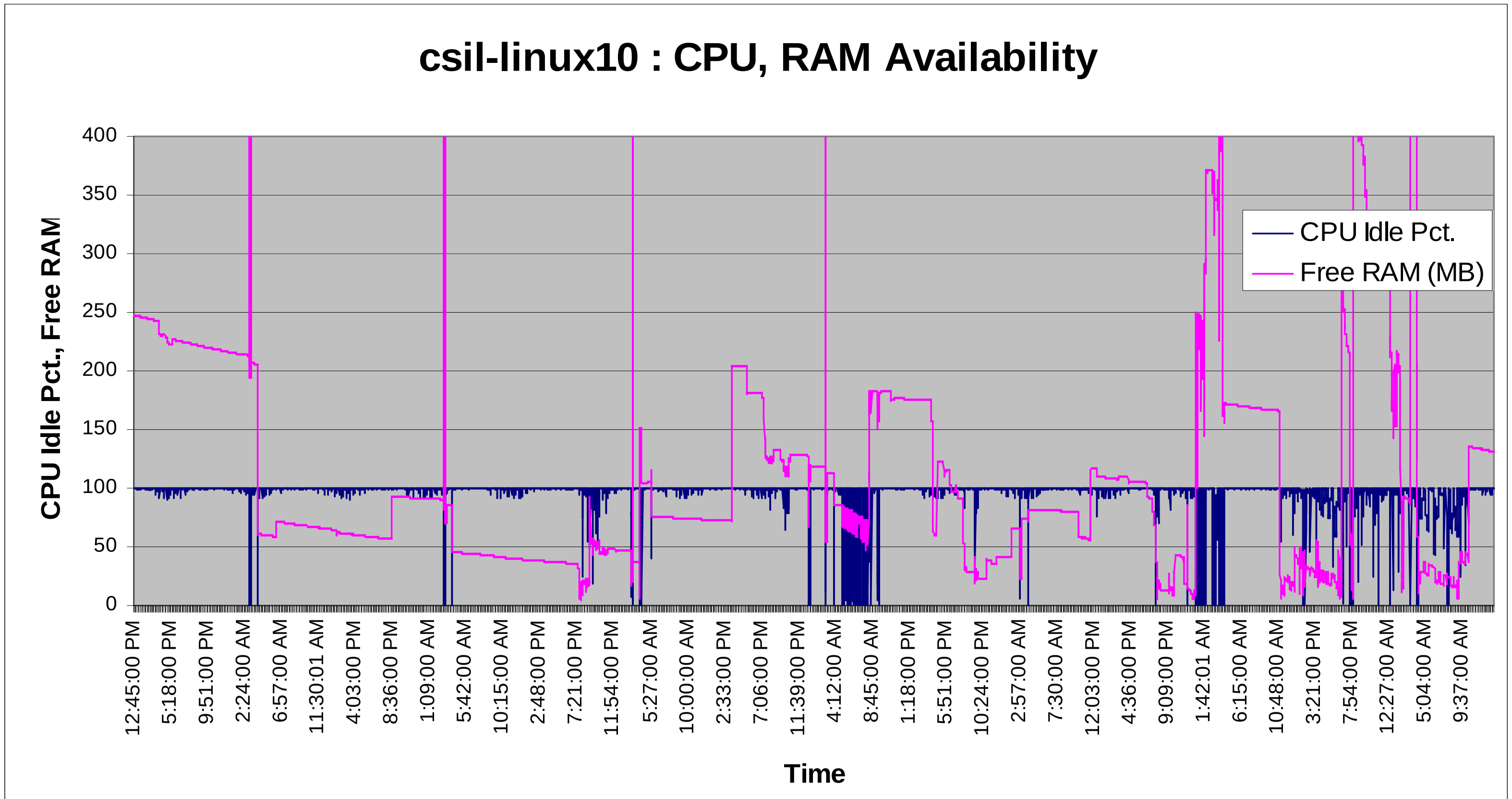


Figure A.6: CPU and Memory traces for csil-linux10.