Autosys: Performance driven automatic system configuration tool

ABSTRACT

With the widespread use of web applications and especially increasing popularity of e-commerce, social networks, web services and cloud computing has introduced a number of research challenges to find the optimum system configuration to support service level agreements. Two most commonly used QoS metrics for web applications are response time for the user and throughput analysis. Web application normally consists of multiple tiers and a request might have to traverse through all the tiers before finishing its processing. As a result a response time for a given request is a sum of response time of all the tiers. In the same way overall throughput is also a function of all the tiers. Since the expected response time at any tier depends upon various software and hardware properties, many different configurations can provide the same QoS requirements. Researchers have shown that finding theoretical optimum configuration to meet QoS and cost requirements is a NP complete problem [8], which leads us to find alternative approaches for solving optimum system configuration problem. In this paper we propose an approach which combines empirical data and case base reasoning techniques to find the optimal system configuration.

Categories and Subject Descriptors

System Configuration [**Case Based Reasoning**]: Cloud Computing

Keywords

System Configuration, Cloud Computing, Case Base Reasoning

1. INTRODUCTION

Increasingly use of Web applications, such as electronic-commerce, social networks, search engines, financial services and Web services, has introduced a number of new research challenges for finding the optimum configuration to meet the QoS requirements. The widespread use of Cloud computing as a system deployment infrastructure has made the problem further complicated. As most of these applications are user oriented, one of the main objectives is to keep their users satisfied by means of meeting certain quality of service guarantees. Two most commonly used quality of service parameters for these applications are a) response time; time taken to process a user request and b) throughput analysis; number of maximum requests that the system can process at any given time.

Apart from the usual advantages of modularity with well defined interfaces, the multi-tier architecture is intended to allow any of the tiers to be upgraded or replaced independently as requirements or technology change. As a result most of the applications nowadays commonly use multi-tiered architecture, where a user's request is handled by multiple tiers (layers) of servers. For example, a typical web-service system can be thought to consist of three tiers: web servers, application servers and database servers. First user sends the request to the Web server and then Web server does the load balancing and forward the request to the application server, and then application server might need to talk to database server to obtain user data or data requested by the user. Within each tier, multiple machines can be provisioned to share the incoming workload and provide additional computational power. As a consequence of this type of configuration, overall system performance depends on a number of different known and unknown parameters, which expands from hardware configurations(number of servers, memory, CPU etc...) and software configurations(thread pools, connection pools, garbage collection etc...). This makes the theoretical analysis of these systems so complicated and finding optimum configuration using theoretical approaches is shown to be a NP complete problem [8].

In this paper we present an approach which combines empirical analysis of multi-tier system and case based reasoning (CBR) approaches to find the optimal configuration to meet the service level agreements and cost requirements. In our empirical system analysis we have used an automated staging analysis tool called Elba [2] [11] [14], and we have collected more than 1000GB of data in over 6,000 experiments. These experiments cover scale-out scenarios with up to 30 servers with different software and hardware configurations. This includes a number of different applications server types (Apache Tomcat and JOnAS), various types of database systems (MySQL and PostgreSQL), different type of environments (Amazon EC2 and Emulab), different types of applications (RuBBoS and RuBiS), various workloads (1000

SAC'10 March 22-26, 2010, Sierre, Switzerland.

clients to 13000 clients) and different types request types (read only and read write mix). Our data set consists both QoS data (response time and throughput) and system data (memory usages, CPU usage, IO usage, network usages and etc...). We apply CBR retrieving algorithms to search our dataset for a given user query and provides the user with a small set of configurations from which user can pick the best match, and then our system retain those information for future retrieving process.

This paper makes two main contributions, firstly it uses an empirical data to provide the optimum configuration to meet the certain service level agreements under certain cost restrictions. Secondly it uses empirical data to provide system bottlenecks for a given configuration (combination of both software and hardware) under certain workload. In the first case we use the Cover tree retrieval [15] mechanism to search the dataset for a given SLA requirements, and in the second case we use concept learning mechanism [16] to find the system bottlenecks for a given configurations. We have implemented two versions of our tool, standalone version and web based version.

The remainder of this paper is structured as follows: Section 2 provides brief background and introduces some of the concepts, and Section 3 outlines experimental setup and methods. Section 4 provides detailed description about our contribution and implementation. In Section 5 we evaluate two retrieval algorithms we have used based on the performance. Related work is summarized in Section 6, and Section 7 concludes the paper.

2. BACKGROUND

This section provides a brief description of techniques and system we have used. In section 2.1 we give a motivating use case where our tool can be used, and next section provides an overview on Elba staging system. In section 2.3 we introduce Amazon EC2 cloud, in last two sections we discuss two retrieving algorithms we have used for our tool.

2.1 Motivating Example

Assume someone wants to start an e-commerce company similar to ebay or amazon, and further assume that he has also developed the initial QoS metrics and he is under a certain budget. Now he needs to find the number of servers he needs to buy to meet the requirements, or he needs to find the number of nodes he needs if he is going to a commercial cloud. To succeed in the business first impression should be the best impression, so he needs to make sure he has all the resources in place to meet the estimated QoS requirements. The main problem is how he finds the system configuration he wants to achieve his goals, one way to do is; he buys small number of servers and then sees whether those can provide his requirements or not. If his selection was not correct his business might be in trouble. Other solution is to buy large number of servers, which will cause for short term (initial budget) and long term cost (power and other maintaining cost). In these scenarios our tool become very useful, he can get very good initial estimate using our tool, which help him to save both his time and money.

2.2 Elba Staging System

The goal of the project is to provide a thorough, low-cost, and an automated approach to staging that overcomes the limitations of manual approaches and recaptures the potential value of staging. In Elba processes there are three major components when automating the staging, i.e., the application, the workload, and finally quality of service requirements. One of the main research challenges is the integrated processing of these different specifications through the automated staging steps. In other words we need to merge Service Level Agreement (SLA), Test Based Language (TBL) and workflow into the automating process and come up with a specification for the application. Specification of the production applications and their execution environments consists of a number of research challenges. First, automated re-map of deployment locations to staging locations. Second create consistent staging results across different experiments and extensibility to many environments and applications.

2.3 Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) [5] is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. Amazon EC2's simple web service interface allows us to obtain and configure capacity with minimal friction. It provides us with complete control of our computing resources and let us run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing us to quickly scale capacity, both up and down, as our computing requirements change. Amazon EC2 changes the economics of computing by allowing us to pay only for capacity that we actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios.

2.4 Cover Tree algorithm

Cover tree is a tree data structure for fast nearest neighbor operations in general n-point metric spaces (where the data set consists of n points) [15]. The data structure requires O(n) space regardless of the metric's structure yet maintains all performance properties of a navigating net [17]. If the point set has a bounded expansion constant c, which is a measure of the intrinsic dimensionality, as defined in [18], the cover tree data structure can be constructed in $O[c^6 n \log n]$ time. Furthermore, nearest neighbor queries require time only logarithmic in n, in particular $O[c^{12} \log n]$ time. A cover tree T on a data set S is a leveled tree where each level is a "cover" for the level beneath it. Each level is indexed by an integer scale i which decreases as the tree is descended. Every node in the tree is associated with a point in S. Each point in S may be associated with multiple nodes in the tree; however, any point appears at most once in every level.

2.5 Exemplar-based Concept Learning Algorithm

Concept Learning is a heuristic approach for classification [16], where this approach differs from the usual concept learning task in three ways. First classification must be explained, second, a program for this task must accommodate incomplete case description, and third, the program must learn domain specific knowledge for inferring case features needed for classifications. The traditional approach to concept learning and classification relies on generalizations. However with exemplar based approach, concepts are learned by retaining exemplars, and new cases are classified by matching them with the exemplars.

3. EXPERIMENTAL ENVIRONMENT

In this section we first we discuss two benchmarks application we have used for our experiments, and then we discuss our experimental infrastructure, software configuration and magnitude of our data set.

3.1 Benchmark Applications

Among N-tier application benchmarks, RUBBoS and RUBiS have been used in numerous research efforts due to its real production system significance. According to our understanding these two applications can mimic the behavior similar to an e-commerce application and social network. As a result our assumption is that the data collected by running these two can be used to find the system configuration for similar type of applications.

RUBBoS [3] is an N-tier e-commerce system modeled on bulletin board news sites similar to Slashdot. The benchmark can be implemented as three-tier (web server, application server, and database server) or four-tier (addition of clustering middleware such as C-JDBC) system. The benchmark places high load on the database tier. The workload consists of 24 different interactions (involving all tiers) such as register user, view story, and post comments. The benchmark includes two kinds of workload modes: browse-only and read/write interaction mixes.

RUBiS [4] is an auction site prototype modeled after eBay.com that is used to evaluate application design patterns and application server's performance scalability. This auction site benchmark implements the core functionality of an auction site: selling, browsing and bidding. It distinguishes between three kinds of user sessions: visitor, buyer, and seller. For a visitor session, users need not register but are only allowed to browse. Buyer and seller sessions require registration. In addition to the functionality provided during visitor sessions, during a buy session users can bid on items and consult a summary of their current bids, rating and comments left by other users. Seller sessions require a fee before a user is allowed to put up an item for sale. An auction starts immediately and lasts typically for no more than a week. The seller can specify a reserve (minimum) price for an item.

3.2 Experimental Infrastructure

We have run a number of experiments over a wide range of configurations and workloads in two main different environments (Amazon EC2 [5] and Emulab [1]) . Table 1 shows the magnitude and complexity of a typical experimentation cycle, and Table 2 summarizes different types of software configuration and hardware configuration we have used for our empirical system testing. For each of the different hardware types and database management system configurations, the total experimental data output averaged at around 277,841,000 one second granularity system metric data points (e.g., network bandwidth and memory utilization) in addition to higher-level monitoring data (e.g., response times and throughput). In the course of conducting our experiments, we have used (both concurrently and consecutively) 91,598 test bed hardware nodes. Our data warehouse is filled with around 3,334,100,000 system metric data points. In order to investigate system behavior as low as SQL query level, we have modified (if necessary) the original source code of all software components to record detailed accounting logs. Because an empirical analysis of the experimental results shows that detailed logging can affect overall performance up to 8.5 percent, we additionally implemented effective sampling algorithms to minimize the logging performance impact. We should mention that all system performance measurements in this paper (i.e., throughput and response time) were collected without such detailed logging turned on. The latter was solely used for specific scenario analysis.

Although the deployment, configuration, execution, and analyzing scripts contain a high degree of similarity, the differences among them are subtle and important due to the dependencies among the varying parameters. Maintaining these scripts by hand is a notoriously expensive and error-prone process. To enable experimentation at this scale, we employed an experimental infrastructure created for the Elba [2] [13] project to automate system configuration management, particularly in the context of N-tier system staging. The Elba approach divides each automated staging iteration into steps such as converting policies into resource assignments, automated code generation, benchmark execution, and analysis of results.

4. IMPLEMENTATION

This section provide overview of our system implementation, first we discuss two different types of our implementation; standalone version and application server version. Then we discuss about implementation details of two algorithms that we have used, and finally we discuss interaction to jColibri framework [6].

4.1 Standalone Implementation

We create standalone version of AutoSys with the basic CBR cycle by using the jCOLIBRI framework [6]. The jCOLIBRI framework is an object-oriented framework in Java for building Case-Based Reasoning (CBR) systems. It supports Textual-CBR, Knowledge Intensive CBR with Description Logics reasoning through Ontologies, Web interfaces, evaluation of the generated applications. A typical CBR cycle includes retrieval, revise, reuse and retain. In the retrieval stage, the customer specifies a configuration according to the configuration attributes. For example, he can specify hardware configurations, software configurations, applications, web configurations, app configurations, cluster configurations and database configurations. After retrieval, the CBR system will return the most similar case to the customer. The solution may include the response time, the throughput and possible bottleneck as well as CPU_utilization and NetWork_utilization. For another example, he can also find whether a given configuration causes system bottlenecks or not. Figure 1 shows a retrieval result for a query containing: read write workload of 2000 with 14 nodes and it show that the system does not have bottlenecks for the given configuration.

4.2 Web Based Implementation

We found that making our data available over the web will be useful for anyone who is interested in finding system configuration for a given SLA. The key assumption is he can decide number of servers he needs to buy as well as type of software configuration he needs to use to meet the SLA using our empirical dataset. His selection based on our data will not be 100% accurate but that provides an initial starting point, or he can find the expected system behavior based on his budget and then adjust the SLA.

Data metrics	Normal MySQL	Small MySQL	Normal PostgreSQL	MySQL Cluster	All	All in paper
# experiments	871	1,352	1,053	416	6,318	42 (0.67%)
# nodes	15,769	18,382	17,693	6656	91,598	1,590 (1.7%)
# data pts.	459.7M	713.4M	555.7M	223.7M	3,334.1M	68.35M (2%)
Data size	82.8 GB	129.3 GB	70.6 GB	40.2 GB	525.6 GB	4.1 GB (0.7%)

Table 1: Data set size and complexity for RUBBoS experiments.

(a) Software setup.

Function	Software
Web server	Apache 2.0.54
Application server	Apache Tomcat 5.5.17
Cluster middleware	C-JDBC 2.0.2
Database server	MySQL 5.0.51a PostgreSQL 8.3.1 MySQL Cluster 6.2.15
Operating system	Redhat FC4 Kernel 2.6.12
System monitor	Systat 7.0.2

(b) Hardware node setup.

Type	Components		
Normal-Emulab	Processor	Xeon 3GHz	64-bit
	Memory	2GB	
	Network	6 x 1Gbps	
	Disk	2 x 146GB	10,000rpm
Small-Emulab	Processor	PIII 600Mhz	32-bit
	Memory	256MB	
	Network	5 x 100Mbps	
	Disk	13GB	7,200rpm
Small-Amazon EC2	Processor	1 EC2 Compute Unit	32-bit
	Memory	1.7 GB	
Lareg-Amazon EC2	Processor	4 EC2 Compute Units	64-bit
	Memory	7.5 GB	

Table 2: Details of the experimental setup.

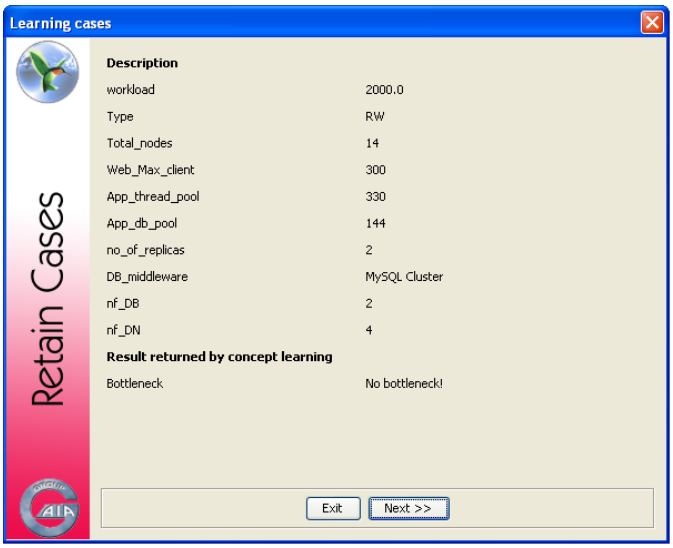

Figure 1: Conect Learning : Standalone implementation

We have implemented our Application server version of AutoSys using Java and Java Server Pages. In our implementation we have incorporated the CBR life cycle (Retrieve, Revise, Retain and Reuse) using cookie based session management; as a result once he enters into the system his credential will be there in the system as long as he has valid cookies. One of the key design goal was to provide a fast response to the users, depending on the type of CBR retrieving algorithms. As we have discussed in the Evaluation section we find Cover Tree based retrieving algorithm outperforms other algorithms as well as has a very good accuracy. At the system startup time we populate and build the case base and then we use it throughout the running time of the system. Our current implementation uses in-memory version of HyperSQL [7] DMBS, however we can easily move to different database management system since we use Apache Hibernate as the ORM.

4.3 Cover Tree implementation

The cover tree is a tree data structure for fast nearest neighbor operations, in general n point metric spaces (where the data set consists of n points). We first pick the first case from the case base and obtain the similarity between this case and the the rest of the cases in the case base. We then divide the other cases into 4 groups by the similarity measure. In those 4 groups, we pick up the first case in each group and then obtain the similarity between selected case and the other cases in the group. We then divide the other cases into 4 groups by the similarity measure. Then we have totally 4^2 groups. Likewise we continue the process until we reach the level 0, and then we have 4^8 nodes in the tree. Thus each leaf node covers one or more cases (cover set).

At the time of search we follow exactly the same procedure as we follow during the tree building, first we clone the tree and then try to insert the query into the cloned tree and find the K nearest neighbors. In our algorithms we have chosen our base as 4 and number of levels to be 8. The main reason behind that is that it would be enough to build a somewhat balanced tree. One of the main important factors behind the algorithm is the distance function. In our case we use similarity function as the distance function. However during the experiments we found that just having the similarity between the cases does not build a balanced tree, so we incorporated a "mod" function to the distance function. As a result, our distance function is combination of similarity function and mod function, and it is a one-way function.

4.4 Concept Learning Implementation

In this algorithm, it tries to classify the case into a specific category. With the system dataset, there are two categories, i.e., bottleneck (CPU utilization => 95%) and no bottleneck (CPU utilization < 95%). It tries to classify the case into the

two categories by using the important features of the categories. It will first learn what the features of the categories are and how important those features are. Then it will use the knowledge to classify the case into the two categories.

For example, we have 561 cases and 423 of them are in the bottlenecks category. From these 423 cases, we count the number of the cases for different features. For instance, for feature "DB node=4", we may have 282 out of 423 cases which have CPU utilization above 95%. Then, we map the specific feature "DB node=4" to 282 cases in the category "bottlenecks". Similarly, we may map the rest to "no bottleneck". That means, if the case has value "DB node=4", it should 66% be classified in the category "bottlenecks" while 33% in the category "no bottlenecks". When the customer gives the value of the features like number of nodes, workloads, max client the algorithm will do the comparison for all the features and give the final classification result. The customer can also modify the value of the features, change the result and retain the cases to "teach" the CBR system.

5. EVALUATION

This section provides evaluation of our two algorithms based on initialization time, query processing time and accuracy.

5.1 Initialization Time

First to collect information about query processing time and initialization time we instrument our program to collect time-stamps information. When collecting those information, we run the initialization process 10 times and then get the average initialization time, and Table 3 shows the initialization time for the two algorithm. It is clear from the Table 3 that the initialization time for the cover tree is comparatively higher, the main reason is in the case of cover tree it has to build the full tree at the initialization time, as the dataset increases time also increases. Because when the number of cases in case base become higher number of comparisons become higher since the time is not linearly correlated with the size. However in concept learning, it scans once so time take to initialize is very low. For the comparison purposes we have also evaluated the default retrieving algorithm in jColibri framework, which outperform the two algorithms when it come to initialization, because it does not involve any comparison work at the initialization time. Here total time is combination of database populating time and case base building time, in both the cases database populating time is same.

Algorithm	Initialization Time (ms)
Cover Tree	180,391
Concept Based	12,032
Jcolibri default	10,718

Table 3: Initialization Time Comparison

5.2 Query processing time (Retrieving time)

Table 4 provides a query processing time for the two algorithms. When we calculate the query processing time; we first select one query and run it for 10 times and then calculated the average processing time. As the Table 4 illustrates cover tree is much faster compared to the default retrieving algorithm in Jcolibri, and which is what we want for a Web application, since user does not want to wait too long to get the response.

Algorithm	Processing Time (ms)
Cover Tree	79.6
Concept Based	46.9
Jcolibri default	255.7

Table 4: Query processing time

5.3 Retrieval Accuracy

We use a "leave-one-out" method (where one case is removed from the case base and used as the query). Subsequently we conducted this for 2000 cases in Cover tree and 3000 cases in jColibri default algorithm and our accuracy values are shown in Table 5. One thing to note here is that in both the cases we select the first five cases and see whether it contains the corresponding description for the query if so we make as correct else it is wrong. In the case of cover tree we have some less accuracy, and main reason is that the data set contains number of similar cases, so cover tree will find most of them, and when we filter 5 cases those cases might not come to the list, however if we increase the number of interested cases then both have the similar accuracy.

According to these results we can claim that cover tree is much suitable for the retrieving algorithm for the web application, since it has a very good accuracy and faster retrieving time. In the case of Concept learning we have acceptable accuracy, and the justification is in our algorithm where we have considered the average CPU to find the bottlenecks, in reality we have multi-bottlenecks and just getting average CPU does not provide the accurate results. As an improvement we need to improve our algorithm to take the density analysis of CPU rather than taking the average.

6. RELATED WORK

There are a bunch of works towards automatic system configuration issues. Kamalika *et al.* [8] discussed the server allocation problem in multi-tier system and in their approach they study computational complexity of the server allocation as a non-linear optimization problem, which they call the multi-tier problem. First they show, for the case of variable number of tiers, the decision version of this problem is NP-Complete. Then they present a simple two-approximation algorithm which runs in linear time and a fully polynomial time approximation scheme. For the case of constant number of tiers, they show that the problem is polynomial time solvable. Their approach is totally based on theoretical background, when it come to practice system are much more complex and it is hard to represent as an equation of set of resources and cost, in our approach we test the system using real application and real hardware and then collect the data, based on those data we can provide a much better solution.

Zhang *et al.* [9] have modeled the multi-tier resource allocation problem as a non-linear integer optimization problem and proposed heuristics to solve it optimally. Zhu *et al.* [10] addressed the issue of allocating resources (machines) in a tree-like topology of a data center, considering performance constraints such as link bandwidth and switch capacity while minimizing communication delay between the

Algorithm	Number Query cases	Correct Cases	Incorrect cases	Percentage (100%)
Cover Tree	200	194	6	97.00%
Jcolibri default	300	298	2	99.33%
Concept (bottlenecks)	247	162	85	65.58%
Concept (no bottlenecks)	78	70	8	89.74%

Table 5: Retrieval Accuracy

assigned servers. They propose a mathematical optimization model with binary variables for optimally configuring the topology. Our work differs also from this, because in our approach we use empirical analysis, and there we have conducted experiments on several different configurations and collected data.

Malkowski *et al.* [12] have discussed about a unique approach to configuration planning based on an iterative and interactive data refinement process. More concretely, their methodology correlates economic goals with sound technical data to derive intuitive domain insights. They have also implemented their methodology as the CloudXplor Tool to provide a proof of concept and exemplify a concrete use case. CloudXplor, which can be modularly embedded in generic resource management frameworks, illustrates the benefits of empirical configuration planning. To some extents our approach and their approach has number of similarities, both the approaches uses the empirical data for the configuration planning. One of the main difference is that we have used well known case base reasoning approach to provide a solution to a given configuration problem. One other difference is our web application is publicly available, so anyone who wants to find a solution to a system configuration problem can use our tool.

7. CONCLUSION

With the widespread use of web application, and especially increasing popularity, e-commerce, social networks, web services and cloud computing have introduced a number of research challenges to find the optimum system configuration to support service level agreements and to meet the quality of service requirements. In the recent research system configuration has become a difficult task due to cloud computing, green resource management challenge. In this paper we have developed a tool to use empirical data and case base reasoning approaches to find the system configuration to meet certain SLA and for a given configuration to find the availability of the bottlenecks and location of the bottlenecks. We have also evaluated our implementation based on retrieving time and accuracy and find that our system perform well in both areas. Our immediate future work is to convert the existing data into our case base, for this paper we have only use less than 10% of the available data. Second, current implementation of the Web application does not support the concept learning, so we need to integrate concept learning into that. Finally, we need to graphically represent the existence of system bottlenecks and optimum configuration.

8. REFERENCES

[1] Emulab - Network Emulation Testbe. `http://www.emulab.net`.

[2] The Elba project. `http://www.cc.gatech.edu/systems/projects/Elba/`.

[3] RUBBoS: Bulletin board benchmark. `http://jmob.objectweb.org/rubbos.html`.

[4] RUBiS: Rice University Bidding System. `http://rubis.ow2.org/`.

[5] Amazon EC2: Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`.

[6] jCOLIBRI: CBR Framework. `http://sourceforge.net/projects/jcolibri-cbr/`.

[7] HyperSQL. `http://sourceforge.net/projects/hypersql/`.

[8] Kamalika Chaudhuri, Anshul Kothari, Ram Swaminathan, Robert Tarjan, Alex Zhang, Yunhong Zhou. Server Allocation Problem for Multi-Tiered Applications. In *Algorithmica*, 2007.

[9] A. Zhang, P. Santos, D. Beyer,& H.-K. Tang. Optimal server resource allocation using an open queueing network model of response time. In *Technical Report HPL-2002-301*, HP Labs, 2002.

[10] X. Zhu, C. Santos, J. Ward, D. Beyer, & S. Singhal. Resource assignment for large-scale computing utilities using mathematical programming In *Technical Report HPL-2003-243R1*, HP Labs, 2003.

[11] Galen Swint, Gueyoung Jung, Calton Pu, Akhil Sahai. Automated Staging for Built-to-Order Application Systems. In *IFIP/IEEE Network Operations and Management Symposium (NOMS 2006)*, April 2006, Vancouver, Canada.

[12] Simon Malkowski, Markus Hedwigy, Deepal Jayasinghe & Calton Pu. CloudXplor: A Tool for Conguration Planning in Clouds Based on Empirical Data. In *ACM SAC*, 2010.

[13] Galen Swint, Calton Pu, Charles Consel, Gueyoung Jung, Akhil Sahai, Wenchang Yan, Younggyun Koh, Qinyi Wu. Clearwater - Extensible, Flexible, Modular Code Generation. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*,November 7-11, 2005. Long Beach, California.

[14] Sahai, Akhil, Calton Pu, Gueyoung Jung, Qinyi Wu, Wenchang Yan, Galen Swint. Towards Automated Deployment of Built-to-Order Systems. In *Proceedings of the 16th IFIP/IEEE Distributed Systems; Operation and Management (DSOM 2005)*, AOctober 24-26, 2005. Barcelona, Spain.

[15] Alina Beygelzimer, Sham Kakade, John Langford. Cover Trees for Nearest Neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006.

[16] B. W. Porter , R. Bareiss , R. C. Holte. Concept learning and heuristic classification in weak-theory domains. In *Artificial Intelligence, v.45 n.1-2, p.229-263*, Pittsburgh, Sep. 1990.

[17] Krauthgamer, R & Lee, J. The black-box complexity of nearest neighbor search. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (pp. 858Ű869)*, 2004.

[18] Karger, D., & Ruhl, M Finding nearest neighbors in growth restricted metrics. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (pp. 741Ű750)*, 2002.