Monitoring Multiple Concurrent Service Level Parameters with Multidimensional Trees

Andreas Kiefer [1], Elias P. Duarte Jr. [1], and Cristina D. Murta [2]

[1] Dept. Informatics, Federal University of Paraná (UFPR)
P.O. Box 19081 Curitiba PR 81.531-980 Brazil
{andreas,elias}@inf.ufpr.br
[2] Department of Computing, CEFET-MG
Belo Horizonte 30.510-000 Brazil
cristina@decom.cefetmg.br

Abstract. The introduction of new computing paradigms in the Internet as well as the increasing size and complexity of services and resources demand the development of new approaches for defining and monitoring service levels. It is often necessary to keep track of multiple concurrent service level requirements. In this paper we present a service level monitoring strategy that allows both online and offline tracking the performance of multiple concurrent resources. Data is collected with SNMP (Simple Network Management Protocol). The strategy is based on building multidimensional search trees. k-d (k-dimensional) trees are employed for online continuous monitoring, and k-d-B trees are employed for offline monitoring, based on logs of monitored data. Searching with the proposed strategy has cost $O(logN)$ where N is the number of samplings or log size. The strategy allows clients and providers to confirm whether contract specifications were hold or not, and for how long. Experimental results are presented, including a comparison of the proposed approach with a traditional database. A practical tool was implemented and results are shown for a set of monitored Web and Video servers, as well as for monitoring data obtained from a real Telecom billing system.

1 Introduction

Cloud computing [1], as well as utility [2] and grid computing [3] have changed the requirements of Internet users. In these systems, services and resources are shared, being provided to diverse customers by several providers. Customers need the assurance of performance and dependability levels, typically specified by service contracts [4, 5]. A service contract is a formal service agreement defined with a Service Level Agreement - SLA [6]. A contract defines the common understanding about the service, obligations, responsibilities, priorities, guarantees, and also the minimum service level that is acceptable for the customer, as well as penalties in case the service levels are not met. These specifications

usually involve several parameters and different combinations of desired levels [4, 5].

A service provider must continuously monitor the system, in order to guarantee that its multiple concurrent customers are being assigned the amount of resources that guarantees previously agreed on service levels. The provider must deal with different customer requirements and must ensure that each customer receives enough resources for each task sent to the system [7]. The provider must evaluate and decide in real time for each task which resources must be assigned for its execution. This is a continuous process: assigning tasks to resources must be evaluated and tasks should be re-scheduled as needed.

The process of selecting the right computational elements that will execute a set of tasks is complex, depending on task priority, the capacity of resources and the system load. Implementing such decision and resource allocation process requires a systematic approach for continuously collecting and organizing a set of performance and system load parameters from all system components (e.g. processors or network connections) in order to match resources and tasks effectively and continuously for each task that arrives at the system. This is a complex management process [8] that involves a very large amount of monitoring data, which must be organized, indexed and queried very frequently.

This work presents an efficient approach for monitoring service level that keeps track of multiple simultaneous parameters. The monitoring information can be used to decide and match resources and tasks, given a set of service level requirements. A system for mapping information obtained from monitoring k different parameters over a k-dimensional space was implemented and tested. Each dimension represents a measure of a quantitative metric related to a parameter of a system resource, e.g. system load, idle processing capacity, amount of free memory or available network bandwidth. The proposed strategy allows multidimensional range search to be performed with cost $O(logN)$, no matter how large k is, where N is the number of samples or the size of the monitored log. The k-d (k dimensional) tree [9] is used for online monitoring and the k-d-B (k dimensional B) tree [10] is used for offline monitoring.

Related work leads to the conclusion that most grid and utility systems collect and store this kind of information in traditional databases [2], [6] and [11]. Applying multidimensional and range search on those systems usually involves complex and potentially slow procedures, and is not feasible especially for on-line monitoring a set of multiple, concurrent system parameters.

The proposed approach was implemented using SNMP (Simple Network Management Protocol) [12] for instrumentation. Experimental results are presented which show the effectiveness of the proposed strategies. Multiple resources are concurrently and permanently monitored and alarms can be adjusted to inform that some resources are reaching thresholds or are not fulfilling expected goals. Experimental results are shown comparing the proposed approach to a traditional database, for up to 8 million data samples and up to 10 different keys, confirming that the database cannot be used for on-line monitoring of multiple parameters. A case study is described which was executed with real

data obtained from a Telecom billing system, as well as for a set of Web and video servers.

The rest of this work is organized as follows. Section 2 presents an overview of related work. In section 3 the data structures employed are specified, as well as the types of queries supported. Section 4 describes the proposed monitoring strategy. Experimental results and the tool we implemented are described in section 5. Section 6 concludes the paper.

2 Service Level Monitoring

Performance and reliability monitoring is one of the basic functionalities of network management systems [13]. Monitoring systems usually involve a cycle consisting of following steps [14]: (1) resource monitoring data is collected; (2) the obtained data is processed in order to discover resource utilization trends and (3) the system is reconfigured, given the conclusions of the previous step. Service management can be seen as a natural evolution of network management, which actually extends traditional performance and availability monitoring strategies to include the concept of *service* as one of the key managed objects. A service can be provided by several types of entities, and the service definition varies widely, examples include the execution of a task that requires intensive processing power or the execution of a query on a database at a service provider. It is common to have service levels specified by Service Level Agreements (SLA). The purpose of service level monitoring systems is to ensure that clients obtain the required service levels while providers optimize resource utilization.

A SLA is composed by [15]: a description of the service to be provided, the specification of performance and reliability levels required; a detailed contingency procedure, associated monitoring and notification procedures; duties and penalties to be applied if the provided service does not meet the required performance levels. Exceptions are also specified, i.e. situations in which the SLA is not applied. A SLA also includes information such as procedures for performance data collection and metric conversion [2]. Typical metrics include response time and bandwidth available.

Monitoring a specific service given the corresponding SLA involves, besides the SLA specification, obtaining information about the service execution, evaluating the service performance and availability, and deciding on future task allocation policies. If the load on servers is heavy, the previously agreed service levels may not be met. In this case the SLA monitoring system must decide, in real time, which agreement should be violated [11].

The amount of information that a service provider has to deal with is huge. This includes information about system resources and services available, their load, client requests, and their requirements as specified in the SLA. All this information has to be processed in real time, allowing tasks to be scheduled in a way to both meet requirements and use resources effectively.

Related work include a number of frameworks and systems proposed recently to monitor and control service level agreements. A SLA monitoring architecture

is presented in [16]. This system includes modules for SLA specification, server discovery, metric definition, service monitoring and the detection of agreement violation. Frameworks with similar functionalities are described in [2], [6] and [11]. In [11] the authors propose contracts specified with XML; they aim at helping in the definition of precise contracts, avoiding the ambiguity that results when natural language is employed. The system provides strict load control in order to avoid the system to get overloaded and, at the same time, maximize the load to optimize system usage.

In all these proposals the authors recognize that it is very important efficiently collect, store and retrieve service management information. In virtually all related work the authors do not fully specify how these data are managed, i.e. they do not define a data structure or format for keeping or searching the monitoring data. In [2], [6] and [11], the authors mention the use of traditional database systems. It is important to highlight that the sheer amount of data that needs be stored by a service management system, the types of search that must be executed in order to match resources satisfying specific criteria in real time can easily represent more than typical database systems are able to handle. In [17], the authors argue that it is not practical to employ a traditional database systems for keeping/searching service monitoring data, where the authors evaluate the high cost of employing SQL queries in databases in this kind of environment. In the next section we present multidimensional data structures that we propose precisely to solve this problem.

3 Multidimensional Search

Several types of applications in diverse fields require the storage and manipulation of large sets of multidimensional data. Example applications span areas such as Image Processing and Geographic Information Systems. Data can be represented as points, lines, rectangles, areas, volumes, among others on the Euclidean space. The attributes of multidimensional data can be represented as an array. The simplest data unit that can be represented is a point in the k-dimensional space, where k represents it set of attributes. Figure 1 shows an example point in a two dimensional plane. Array `(3,5)` has value 3 for the `x` axis and 5 for the `y` axis.

In the past twenty years numerous data structures have been proposed for keeping and searching multidimensional data efficiently. Most of them are variations of multidimensional search trees or k-d trees (k-dimensional trees) [9, 18]. The k-d tree is a generalization of a binary tree. The k-d tree receives a set of multidimensional points as input data. These points are embedded in a multidimensional space. As a point is inserted in a k-d tree, it causes a recursive space partition based on the division of hyperplans with k-dimensions. For example, if k=3, then the division hyperplanes will alternate among `x`, `y` and `z` axes.

The hyperplanes can be obtained using the mean value of all points in each dimension, which is employed as a *discriminant*. In this way, each other point with the corresponding key value less than or equal to the discriminant is placed

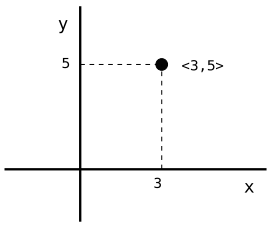

Fig. 1. A point represented in the bidimensional space.

on the left subtree and points with greater values are placed on the right subtree. For each dimension a new discriminant is obtained and the data insertion algorithm is executed recursively until all points have been inserted.

Figure 2 shows two common representations of a two-dimensional k-d tree: on a 2-dimensional space and as an abstract tree. In both representations point `p1` was chosen as discriminant and it divides the space in two hyperplanes. `p1` is the root of the tree. The next division plan crosses `p2` (left subtree) and `p3` (right subtree). This procedure completes when all points have been inserted in the tree.

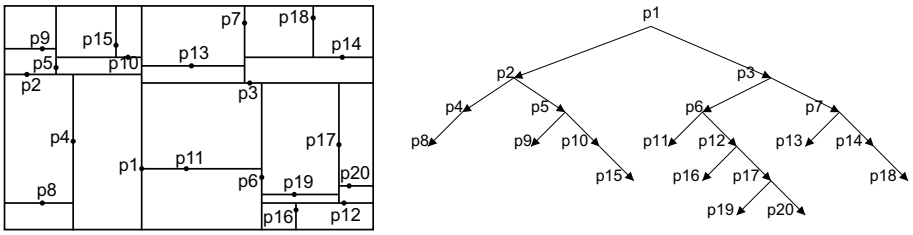

Fig. 2. k-d tree: two-dimensional plane (left) and abstract tree (right) representations.

The k-d tree is a data structure that has to be entirely kept in main memory. In case the tree is too large and must be stored in secondary memory, algorithms will require paging, i.e. obtaining tree chunks from disk. As the tree grows increasingly unbalanced, performance will deteriorate, and logarithmic algorithms may become linear. An alternative to solve this problem is to employ another balanced multidimensional tree meant for secondary memory, that is the k-d-B tree proposed by Robison [10].

The k-d-B tree can be seen as variation of the k-d tree that combines the logarithmic algorithms of k-d tree with the efficient secondary memory management algorithms of B trees. Thus it is efficient to process a tree that is not completely loaded to main memory. The k-d-B tree is a multi-way tree that keeps all leaves at the same level. Like the k-d tree, the k-d-B tree also partitions the multidi-

mensional space on which data are embedded in sub-spaces. Each internal node corresponds to a rectangular region and its children define a disjoint partition of that region. The points are stored in the leaves of the tree.

Figure 3 shows an example k-d-B tree. Each internal node represents a rectangular region in the planar representation, and each leaf represents a partition obtained by recursive decomposition of the `x` and `y` axes using parallel lines (`x1`, `x2`, `x3` and `x4`; `y1`, `y2` and `y3`).

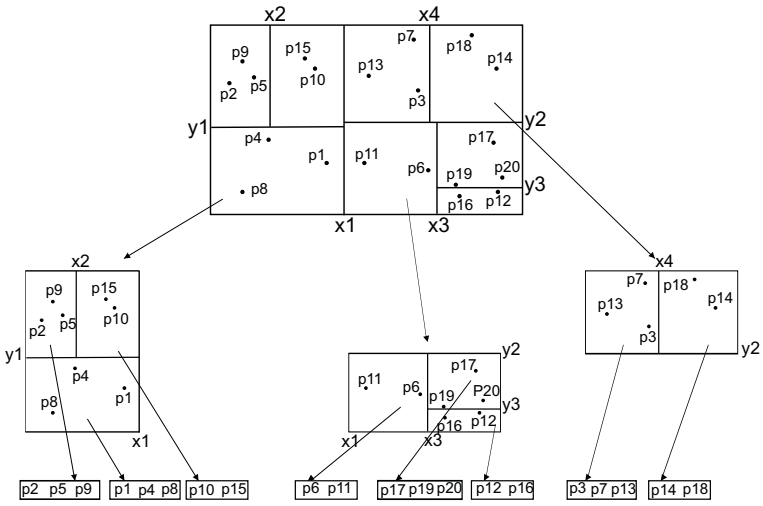

Fig. 3. A k-d-B tree planar representation.

3.1 Multidimensional Search Types

Searches are typically initiated in response to a query. A search algorithm returns the records that match the query condition. A multidimensional search, sometimes also referred as *associative search* or *search using secondary keys* operates on records with several keys specified as tuples of attributes. A record can be represented in the Euclidean space as a point, its attributes correspond to the point's coordinates in the space. A tuple represents coordinates in a multidimensional space, where each dimension represents one attribute.

A query retrieves all records that satisfy some properties. Various types of queries exist [19] and are classified according to specification of desired records, also called a *region*. A query always retrieves all records that fall within the specified region. A region is defined as a set of maximum and minimum coordinate values in the geometric space. A description of the main types of queries follows. The *exact match* is the most simple query, and returns a specific record defined by its k keys. A *partial match* yields a set of registers whose k keys match at least

one key specified in the query. The *range match* looks for all records that have their k keys within the specified ranges of keys of key values. Finally, a *proximity* query looks for the records that are closest to a given set of key values.

4 The Monitoring Strategy

The proposed monitoring strategy is described in this section. Multiple concurrent management objects provide automated data collection of selected system resources, for example, CPU, disk, memory, network interfaces. The monitoring strategy is composed of three phases: data sampling, data insertion into the multidimensional space, and requirements evaluation (assessment of requirements).

The monitored parameters are SNMP objects. These objects collect performance data and quantitative information of the computing device monitored. Each object is associated with requirements described by arithmetic and logic functions. Multiple requirements reflect restrictions on the values of the objects.

To carry out the first phase, sampling, we define the monitoring objects, the SNMP agents from which objects are derived, as well the sampling interval. In the second phase, the sampled values are mapped to a multidimensional space. In this space, each dimension corresponds to an object. Each point of this space is a managed computing resource or device.

Figure 4 shows an example of a bidimensional space. Four computing units are monitored and mapped in a bidimensional space that represents two computing resources: available memory and processor utilization. In this picture, `C1`, `C2`, `C3`, and `C4` represent the resources monitored. The available memory is plotted on the x axis, while the processor utilization is represented on the y axis. The points `M1`, `M2`, `M3`, and `M4` are values for available memory (%) for each of the four resources `C1`, `C2`, `C3`, and `C4`. In the same way, `P1`, `P2`, `P3`, and `P4` are the processor utilization (%) of each computing resource.

The mapping of the objects in the multidimensional plane is done with multidimensional search trees in the second phase. The k-d tree is the data structure chosen for online monitoring, while the k-d-B tree is provided for offline monitoring. The search in both trees can be based on specified computing resource metrics.

In the third phase, the monitoring strategy retrieves data from the updated multidimensional search tree. The queries are built on the requirements. The multidimensional search is applied to all monitored resources.

The multidimensional search returns the devices that match the specified performance parameters and based on thresholds a warming or alarm can be issued if requirements are not met. The tool can be also record the time interval in which resources presented a given performance. A timestamp is appended to the collected data items, so it is possible to evaluate the history of the resource behavior.

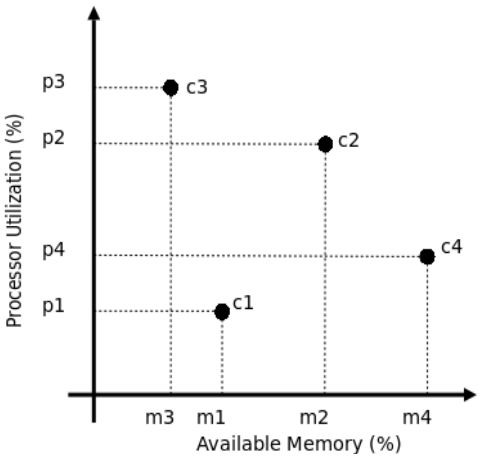

Fig. 4. Mapping of objects and resources in a bidimensional space.

5 Monitoring Tool Description and Experimental Results

In this section we describe a tool that implements the proposed strategy and the experimental setup and results. The tool is composed by a monitoring module which is based on SNMP, and a module to manage the multidimensional trees.

5.1 Description of the Monitoring Tool

The tool was written in C++ and runs on the Linux operating system. The data collection module is based on Net-SNMP[20]. The tool obtains data from SNMP agents. Some SNMP objects are directly used, without any extra processing. Other SNMP objects, e.g. counters, require two sequential samples to be obtained in order to give meaningful information.

Consider for instance how the processor load and memory usage are monitored. The processor load is computed from objects *ssCpuRawNice*, *ssCpuRawUser*, *ssCpuRawSystem*, and *ssCpuRawIdle*, availabled in UCD-SNMP MIB [20]. The values represent, respectively, the time the system was in "low priority", " user", "system", and "idle" modes, all of which are kernel measures.

Memory usage is computed from objects *hrStorageSize* and *hrStorageUsed* implemented in the *Host Resources* MIB [21]. These objects represent, respectively, the amount of available memory and used memory. All samples collected are stored in secondary memory before being handled to the multidimensional information management module, described below.

The multidimensional information management module is responsible for the insertion the collected samples in multidimensional search trees. The k-d-B tree was implemented using the toolkit TPIE (Transparent Parallel I/O Environment) [22], that allows the implementation of external memory algorithms, and minimizes the input and output communication (I/O) performed when solving problems on very large data sets. The k-d tree was implemented using the template libkdtree [23]. Both the k-d-B tree and the k-d tree implementations allow data to have, at least in theory, an unlimited number of dimensions.

5.2 Results

We initially present results comparing the proposed approach with a traditional database, in this case MySQL. We measured the time required for inserting and searching up to 8 million monitoring samples. The comparison was performed for off-line monitoring, and thus only with k-d-B trees, which also involve disk accesses. It is important to remark that k-d trees used for on-line monitoring is orders of magnitude faster than both databases and k-d-B trees. In our experiment the database either presented a very high search time or a very high index building time. In order to get roughly the same search time of multidimensional trees (ranging from 2 to 6 milliseconds) the database had to be configured to build indices for all 10 keys. Figure 5 shows the insertion time measured in this case. While in our approach the time required is at most a few seconds, the database required more than 2 hours for fully indexing 8 million records.

We now present experimental results obtained from monitoring three Web servers and three video servers. In the experiments, six parameters were sampled: the time instant the SNMP query was issued; the utilization of the input network interface; the utilization of the output network interface; memory usage; processor utilization; and number of active TCP connections. Servers were connected to a non-dedicated 100Mbps Ethernet network.

The monitoring samples were inserted into the k-d tree. Three experiments were then executed with different purposes. The first experiment was meant to show that the tool performed as expected and that the multidimensional search returned correct values. The second experiment was performed to show that the variation in the number of key attributes did not affect the tool performance. The third and final experiment show the relationship between the response time of the multidimensional search and the number of records stored in the k-d tree. The response time is an important parameter to allow the tool to be used for continuous on-line monitoring.

In the first experiment, a video server was monitored for an interval of 60 seconds. Samples containing the six monitoring parameters were collected every second and inserted in the k-d tree. Figure 6 shows results. Several multidimensional range queries were executed including (ranges are specified in brackets): processor utilization: [5-7] %, output network interface utilization: [15-25] %, memory usage: [85-95] %. For the other parameters, the range of query was the range of the field of records. The multidimensional search is shown in Figure 6. The horizontal lines show the specified limits for the three parameters.

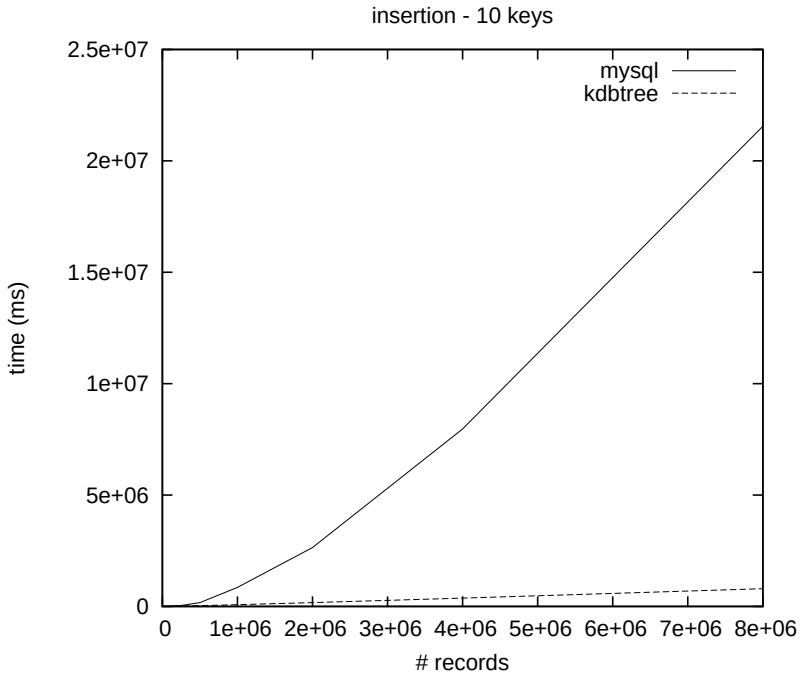

Fig. 5. Comparison of the insertion time: k-d-B tree and MySQL.

The query retrieved 38 records that meet the criteria described above, which means that during approximately half of the time of observation, the selected parameters presented their values within the range specified by the search. This result confirms that the tool produces accurate results as expected, i.e. it allows monitoring and range search for system parameters according to specified values.

The goal of the second experiment is to evaluate the relationship between the number of query attributes, that is, the search keys, and the response time of the search in the k-d tree. The video server and the Web server were monitored. The monitoring phase has produced 600 records with 4 keys, 600 records with 5 keys, and 600 records with 6 keys. A tree was built for each set of 600 records and the insertion time was measured. The experiment was repeated hundreds of times. Table 1 presents the average insertion time and the average search time measured in this experiment with a varying number of keys. It is easy to see that the insertion time is constant, i.e. does not vary with the number of search keys. This result is expected as the multidimensional search tree is a generalization of the binary search tree. At each tree level, one key is used as the search parameter, and guides the traversal. Another consideration is that the time to search the tree does not depend on the type of information stored, i.e. it is independent of the content of records and, consequently, independent of

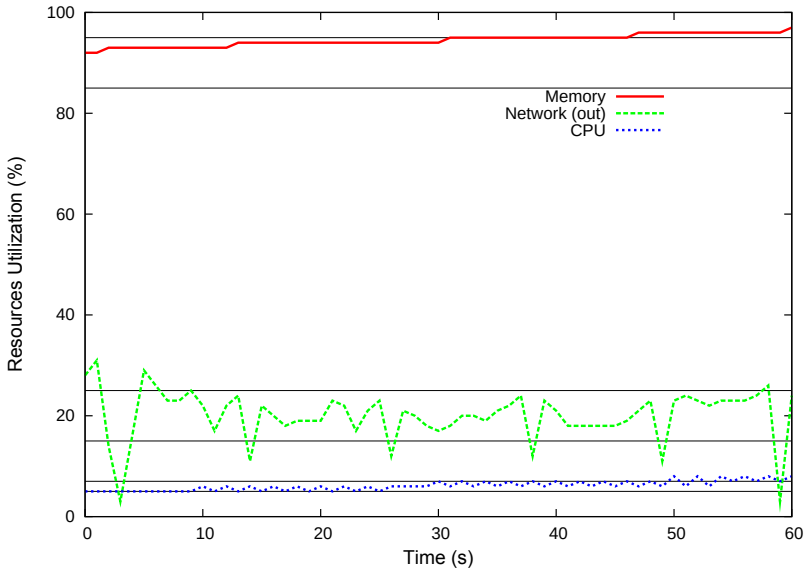

Fig. 6. Utilization of the video server resources.

the kind of monitored resources. The result of this experiment confirms that we are able to issue queries using a large number of keys. As a consequence, we can record and retrieve several system resources simultaneously with no penalties for keeping the chosen data structure.

# keys	# records inserted	Average time insertion (ms)	# records retrieved searched	Average time search (ms)
4	600	20	600	20
5	600	20	600	20
6	600	20	600	20

Table 1. Average time of insertion and search in a k-d tree using 4 to 6 search keys.

The third experiment is designed to evaluate the search time in the multidimensional search tree as a function of the number of records entered and obtained from the experiment. In this experiment, the servers were monitored by a period of 30 minutes (1,800 seconds), and samples were collected every second. These samples were inserted into the multidimensional search tree every 60 seconds. This procedure was repeated until the end of the experiment. For each insertion the time (in ms) of the operation was measured, as well as the time (in

ms) to execute the range search. The range was defined in a way to allow the retrieval of all data stored, which is the most comprehensive search possible.

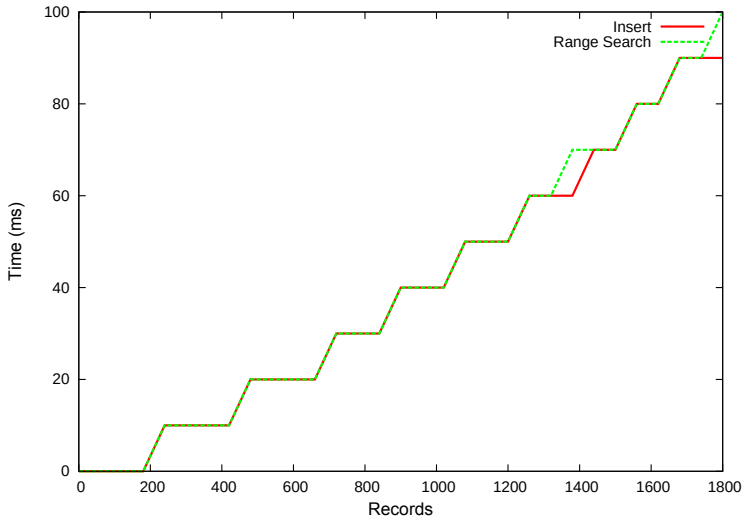

Fig. 7. Insertion and search response times as a function of the number of records in tree.

The response times for the insertion of samples collected from the start of monitoring (generating a history of monitoring), and the response times to search the multidimensional trees with a varying number of records inserted are shown in Figure 7. This Figure shows that both the insertion time and the response time grow with the number of records. For example, when we have 230 records, the response time for the insertion operation is around 10 ms. In order to give an idea of the time limits presented for on-line monitoring we can add the insertion time plus the multidimensional range search time, we obtain a response time of 20 ms. In this case, the proposed strategy takes 20 ms on average to insert 230 records in a multidimensional search tree and to get the result of a query in this data structure. Consider a new request that arrives and has to assigned to a computational unit, selected in terms of performance parameters. For each new request, a search is executed in about 10 ms. For 100 requests, the response time of the search would is about 1 second, which results in a rate of approximately 100 requests answered by second. The response time of the query by interval may be reduced if the number of records stored in the tree is also reduced, this can be done by entering only the most recent records in the tree, which will reflect the recent state of the system.

We also evaluate the proposed tool using Call Detail Records (CDR) of a real telecom billing system. The call processing system evaluated produces an

average load of 300 CDRs/s. Each CDR generates 1KB of data. The system under evaluation processes an average of 60,000 calls/s, and the CDR are collected at every three minutes. We have collected 100,000 CDR, that accounts for approximately five minutes of system activity. We inserted these records in the multidimensional data structure under test. The results shows that the insertion time as well as range search are logarithmic, as expected. This experiment confirms that the data structure can be effectively used to handle real-world systems data.

6 Conclusion

In this paper we presented a strategy to efficiently monitor multidimensional data which consists of several parameters. Data instrumentation is based on SNMP. Queries can include multiple concurrent requirements on several system devices. The multidimensional search can be used for instance to find resources that meet specific requirements within a set of monitored distributed systems. The proposed strategy is based on storing data in multidimensional trees which allow logarithmic range search. A implementation was described that allows both continuous online monitoring with the k-d tree and offline monitoring massive amounts of logged data with a k-d-B tree.

Experimental results are shown comparing the proposed approach to a traditional database. The comparison involved up to 8 million data samples and up to 10 different keys, confirming that the database either requires a time consuming full index creation or presents very high search delays. A set of Web and video servers was monitored. Server resource information was collected and inserted in the data structures, allowing the identification of periods in which the servers were heavily/lightly used during the period of observation. Overall the results confirm that the proposed strategy can be effectively applied to monitor real systems. Besides including the development of a Web interface for the tool, future work is focused on integrating the proposed tool to a cloud computing platform.

Acknowledgements

The comparison of our approach to a traditional database (MySQL) was possible thanks to the implementation and careful evaluation conducted by Saulo Quinteiro dos Santos, Renato Yamazaki, and Luiz F. A. de Prá, at UFPR.
This work was partially supported by grant 311221/2006-8 from the Brazilian Research Agency (CNPq).

References

1. Lawton, G.: Moving the OS to the Web. Computer **41**(3) (2008) 16–19

2. Buco, M.J., Chang, R.N., Luan, L.Z., Ward, C., Wolf, J.L., Yu, P.S.: Utility Computing SLA Management based upon Business Objectives. IBM Systems Journal **43**(1) (2004) 159–178
3. Livny, M., Raman, R.: Enterprise Resource Management: Applications in Research and Industry. In Foster, I., Kesselman, C., eds.: The Grid. 2nd edn. Morgan Kaufmann (2003)
4. Kumar, V., Schwan, K., Iyer, S., Chen, Y., Sahai, A.: The state-space approach to SLA-based management, 11th IEEE/IFIP NOMS (2008)
5. Sall, M., Bartolini, C.: Management by Contract, 9th IEEE/IFIP NOMS (2004)
6. Bouillet, E., Mitra, D., G.Ramakrishnan, K.: The Structure and Management of Service Level Agreements in Networks. IEEE Journal on Selected Areas in Communications, **20**(4) (May 2002) 691–699
7. Abraho, B., Almeida, V., Almeida, J.: Self-Adaptive SLA-driven Capacity Management for Internet Services, 17th IEEE/IFIP DSOM (2006)
8. Taylor, R., Tofts, C.: Death by a Thousand SLAs: A Short Story of Commercial Suicide Pacts. (2006)
9. Bentley, J.L.: Multidimensional Binary Search Trees Used For Associative Searching. Communications of the ACM **18**(9) (1975) 509–517
10. Robinson, J.T.: The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. In Lien, Y.E., ed.: Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, April 29 - May 1, 1981, ACM Press (1981) 10–18
11. Leff, A., Rayfield, J.T., Dias, D.M.: Service-Level Agreements and Commercial Grids. IEEE Internet Computing (July-August 2003) 44–50
12. Harrington, D., Presuhn, R., Wijnen, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks (2002) *Request for Comments 3411.*
13. Stallings, W.: SNMP, SNMPv2, SNMPv3 and RMON1 and 2. 3rd edn. Addison-Wesley Longman Publishing Co., Inc. (1998)
14. Leinwand, A., Conroy, K.F.: Network Management a Pratical Perspective. 2nd edn. Addison-Wesley Longman Publishing Co., Inc. (1996)
15. Verma, D.: Supporting Service Level Agreements on IP Networks. Macmillan Technical Publishing (1999)
16. Molina-Jimenez, C., Shrivastava, S., Crowcroft, J., Gevros, P.: On the Monitoring of Contractual Service Level Agreements (2004) Technical Report series CS-TR-835 April 2004 School of Computing Science, University of Newcastle upon Tyne.
17. Dinda, P.A., Lu, D.: Nondeterministic Queries in a Relational Grid Information Service. In: SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, Washington, DC, USA, IEEE Computer Society (2003) 12–26
18. Bentley, J.L.: Multidimensional Binary Search in Database Applications. IEEE Transactions on Software Engineering **4**(5) (1979) 333–340
19. Gaede, V., Gunther, O.: Survey on Multidimensional Access Methods (Revised Version). ACM Computing Surveys **30**(2) (june 1998) 170–231
20. : The NET-SNMP Project Home Page (2003) `http://net-snmp.sourceforge.net`, Accessed in July 2009.
21. Waldbusser, S., Grillo, P.: Host Resources MIB (2000) *Request for Comments 2790.*
22. : A Transparent Parallel I/O Environment (2003) `http://www.cs.duke.edu/TPIE/`, Accessed in July 2009.
23. : The libkdtree++ Project (2004) `http://freshmeat.net/projects/libkdtree/`, Accessed in July 2009.