Results in Data Mining for Adaptive System Management

Arno J. Knobbe, Bart Marseille, Otto Moerbeek, Daniël M.G. van der Wallen

syllogic

Postbus 2729
3800 GG Amersfoort
The Netherlands
{a.knobbe , b.marseille, o.moerbeek, d.van.der.wallen}@syllogic.com

Abstract

This paper describes the results of using Adaptive System Management on a large-scale site. We describe how large amounts of data, gathered by monitoring complex computer systems, are analyzed and present the results. The results contain useful and unexpected suggestions for improving the effectiveness of managed applications and for reducing the support effort. We also describe a number of different Machine Learning techniques and implementations that were used for the application of Adaptive System Management.

Introduction

As computer networks become bigger and bigger, it becomes more difficult to manage the complex interaction between the different components of the system, such as host, devices, databases and applications. Each of these components can affect the operation of the other components, and ultimately the performance of the tasks a user wants to perform.

Some events that cause a degradation of the performance in a computer system are simple and can easily be solved or prevented by a human system manager. However, with the increasing complexity of modern computer systems, it becomes quite hard to pinpoint the exact cause of the degradation. In most cases bad performance is not even caused by a single, sudden event, but by a subtle interplay between several parameters that causes a gradual change in performance. System management becomes a complex optimization problem instead of a set of simple reactive operations.

It is quite hard for human beings to get clear insight into the operation of all components of a complex computer system. At the same time it becomes more and more easy to record large quantities of data about the different components in a computer system with the current development of large system and network management packages that are aimed at monitoring the system. The application of Data Mining techniques to these complex system management problems [2, 6, 7, 10, **Error! Reference source not found.**], called Adaptive System Management, seems an obvious step. We implemented and applied these ideas and extended earlier reported approaches [10] with first order techniques.

Adaptive System Management

The first requirement for the application of ASM is a collection of data describing the state of the system at regular intervals. Another very important issue is the availability of a measurable performance measure, in system management terms: Service Level Agreement (SLA). ASM is mainly based on analyzing the influence of different system components on this performance measure.

The data that is to be analyzed is produced by a set of *monitors*. Each monitor may be a value obtained by a particular system management system, the result of some script, or simply a value such as the time of day, etc. Each monitor effectively corresponds to an attribute in the data set that is produced. We assume that the monitors produce a vector of measurements periodically, to reflect the situation at a particular moment, or over the interval between the previous and current measurement. Furthermore we assume that we are interested in the dependencies between a single target monitor and the remaining monitors. This can be easily generalized to multiple targets.

If we consider each set of monitor values at a particular moment as an example of some relation between the monitors and the SLA, we can apply Machine Learning algorithms to generate approximations or partial explanations of this relation. In the next section we will consider a number of possible approaches that produce different types of knowledge. Some techniques attempt to find complete explanations, such as *decision tree induction*, some techniques only discover isolated facts about part of the data (nuggets), such as *association rules*. These techniques share a propositional nature. Another approach is *Inductive Logic Programming* (*ILP), a* first order technique which is able to use background knowledge in the learning process.

The data set that is produced by the monitors can be used directly by propositional learning algorithms [10]. However, a lot more background knowledge about the structure of the domain is available, that cannot be used to improve the results of propositional algorithms, such as the ones mentioned above. We are using first order techniques, such as *ILP*, to incorporate this domain information in the analysis, and build on top of existing knowledge about the workings of the system.

The domain information that is used in our ASM environment consists of a collection of relations between different components in the system and the monitors that are defined on them. The hierarchical description of components consists for example of the following logical facts: an AIX4-r1server is an AIXserver is a UNIX server etc. An example is shown in Figure 1.

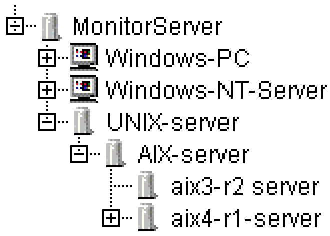

Figure 1: hierarchical modelcomponents

The description of the model consists of relations between components. The relations between components are for example: a disk is a part of a machine, a machine is part of an application etc. An example is displayed in Figure 2.

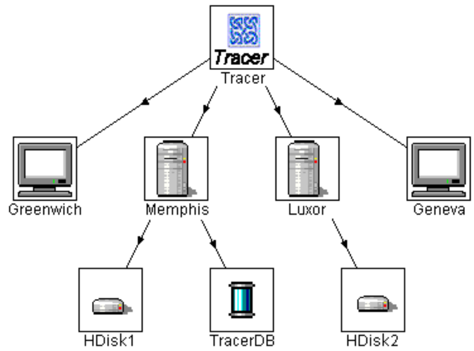

Figure 2: relational model

Techniques

A range of Machine Learning techniques can be applied to the ASM problem. Different techniques produce different results, and each have their specific advantages and disadvantages. Which technique to use depends on the specific needs and interests of the system administrator who is analyzing the system. We distinguish between propositional techniques and first order techniques that allow the inclusion of model information.

Propositional

Decision Trees

Decision trees combine the advantages of good predictive and modeling power with the advantages of a hierarchical organization of the discovered dependencies. By starting at the root of the tree we can examine important dependencies, and go into detail by proceeding towards the leafs of the tree. From a range of decision tree algorithms we will consider C4.5 [14].

Although decision trees can be very useful in analyzing for example the causes of bad performance in a network, they suffer from an important problem that may cause particular dependencies to be overlooked. In most cases, monitors not only provide information about the value of the target, but also about other monitors. In fact monitors may share information about the target. If a particular monitor is used as a split in the decision tree, all information that is shared with another monitor is used, and the other monitors will be "overshadowed" [9, 10].

Top N

The "overshadowing" problem is quite clear when two monitors are used of which one is some increasing function of the other. Unfortunately this is quite often the case for ASM. It may therefore be useful in most cases to not only produce a decision tree, but also a list of monitors ordered by some measure of similarity, for example by the information gain of the best split on each monitor. In a way this represents the available options at the root of the tree, the best of which is selected as the first split. Also rule-induction techniques may be used to produce overlapping rules that will show multiple dependencies. Also many monitors can be expressed as arithmetic functions of other monitors. Especially roughly linear dependencies between monitors are quite common. Simple statistical techniques such as correlation and linear regression can also be very effective to produce a top N.

Association Rules

In many cases monitors represent binary information, for example the availability of a service or application. Association rules are ideally suited to express dependencies between sets of these binary monitors [3, 5, 8]. Even if monitors represent nominal values, a set of binary monitors can be used to represent all possible values of the original monitor. Monitors that have numeric values, such as the response time to a request, can be discretised first.

First order

Inductive logic programming

Inductive logic programming (ILP) [4 , 11, 12, 13, 15] offers a 'knowledge rich' approach to machine learning: ILP can incorporate domain knowledge into the modelling process in the form of a 'logic program' which specifies the links and relationships already known to be in the domain of application, that is the background knowledge. In the domain of ASM this background knowledge consists of the relational model and hierarchical decomposition of components.

A shortcoming of the current Data Mining tools as used in the current ASM tools is the feature based approach (propositional modelling). This gives results similar to: *if monitorY on machine X > 5 then Performance is low.* These kinds of results need extensive interpretation. We need to generalize these specific results ourselves. Are there also other machines for which this rule holds? Such questions can of course be answered by interpreting the result with a system manager. On the other hand, we could use ILP to automate part of this interpretation process. To do this we need to supply the ILP-implementation with information about the system. This background information can be stated in first order logic, in our case the background information consists of an hierarchical description of components and a model of how these components are related.

This type of information about the system can be used by the ILP implementations to generalize about results. The algorithms not only consider individual monitors like *diskio of HDisk1 on Luxor,* but can also find results with a higher level of abstraction like *diskio of a disk on an AIXServer.* Such results are obviously more interesting for system managers, the ILP-algorithms "speak" more their language than the propositional algorithms.

Systems

In our experiments, we have used 3 systems to analyse the data that was stored: the Syllogic Adaptive System Management Tool, Progol from the University of York and Tilde from the University of Leuven.

Syllogic Adaptive System Management Tool

The Syllogic Adaptive System Management Tool is an integrated toolbox that consists of several modules for data collection, data storage, system modelling, data analysis and presentation. All functionality is accessible from a single GUI. The analysis module consists of the propositional techniques described in the previous section. The results of these techniques are presented visually to the user. In the system modelling module the user can define the structure of the system to be analyzed, by visually defining relations between components. Examples are given in Figure 1 and Figure 2.

An important problem to be solved in an Adaptive System Management environment is the collection of monitor data. Collecting the data centrally may be a very expensive operation, resulting in a lot of network-traffic, which in turn influences the overal performance of the system. The Syllogic ASM tool uses a specific architecture for collecting data. The basic component of the data collection system is a collection object. Each collection object performs the same action: retrieve data from the system, which can be another collection object, store the data in memory, file or database and potentially propagate data to parent collection objects. At the leaves of the tree the collection objects retrieve data by taking measurements (e.g. by executing programs) or reading existing statistics (e.g. SNMP or kernel statistics). At the internal nodes of the tree collection objects retrieve data by collecting data from the collection objects that are below them.
By choosing which data to propagate up and which data to store at a collection object, it is possible to find an effective optimization between use of resources (network, storage, computational power) used at collection time and query time. Furthermore, by organizing the tree to minimize dependencies, the system will continue to operate, while there may be a problem in one or more parts of the system.

Progol

Progol is an ILP system that uses a rule-based algorithm. It searches for rules that cover as much of the data as possible. These rules can overlap in the data they cover. Progol [13] was developed at Oxford University Computing Laboratory. Progol can use intensional background knowledge which allows learned clauses to be incorporated for use in later learning. It constructs a pruned top-down search which produces guaranteed optimally short clauses. Furthermore, it supports numerical hypotheses using built-in functions as background knowledge.

Tilde

Tilde [4] is a recent ILP system that builds binary decision trees, having no overlapping coverage. It is derived from Quinlan's C4.5, in that Tilde uses the same heuristics and search strategy. However, it works with a first order representation. Hence, it has C4.5 (when working with binary attributes) as a special case. Because of the use of the divide-and-conquer strategy underlying decision tree technology, Tilde has also proven to be very efficient and to run on large datasets. Other features of Tilde include the ability to handle numbers and to perform regression.

Experiments

This experiment was done at a large site of an airline company. In this experiment we monitored a spare part tracking and tracing system for aircraft. This application uses several IBM AIX servers, an Oracle database, and Windows PC clients in Amsterdam, Singapore and New York. In total 250 monitors were installed on the components of the system. These monitors were implemented on the components and measured among others for example *cpu load*, *free memory*, *database reads* and *nfs activity*. The monitors were executed every 15 minutes and stored in a database. We effictively monitored this system for 2 months, resulting in a table of 3500 timeslices of 250 monitors.

The performance was measured by simulating a task of the application; querying a database, and recording the access time of this database query. The performance measure (SLA) is defined and used as a normal monitor, it only differs in the way it is used in the analysis. All monitors, including the *SLA*, were measured at the same interval. The Service Level Agreement was defined as an operation on this database access time, namely *database access time < 3.* This comes down to an application-user never having to wait more than three seconds for the answer when performing this specific task. This task was seen to be typical for this business system.

Furthermore, a model of the application and the system it runs on was created. The model consists of about 140 components, such as servers, printers, disks and databases. All components that are monitored are in the model as well as the known relations between the components. The relations are one-to-many part-of relations. For an impression of the model see Figure 3.

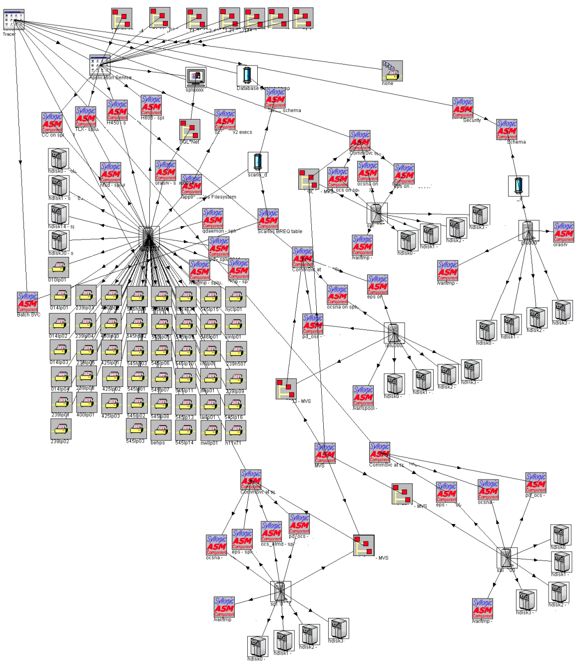

Figure 3: Business system model

Decision Tree

The tree in Figure 4 shows the decision tree induced from the complete dataset. The most important attribute is *paging space used on server 11*. The tree states that a high value for the monitor *paging space used* is the main indicator for bad performance. The amount of used paging space can not be influenced directly, but can be reduced by increasing the amount of physical memory, or limiting the number of applications that run on this server.
The next split on *CPU idle time on server 11* gives additional evidence for the fact that server 11 needs additional hardware or less intensive usage.
The attribute *filetime* measures the delay on scheduled jobs. The application updates request from a mainframe on fixed times. The requests are sent (in batch) from the mainframe to the UNIX-server server 11, and then processed by the application. The split indicates that if the mainframe sends the file more than 2.5 minutes late, the performances drops. This can have multiple causes. Firstly, the network could be down causing the mainframe to fail when trying to send the requests and at the same time causing the performance-measure to time out because the database query is performed over the network. Secondly, the application wastes CPU cycles trying to retrieve a file which is not yet there, because the mainframe application did not yet put it there, which happened in more than 50% of the cases. So, this split indicates that the way the application recieves mainframe requests should be improved.

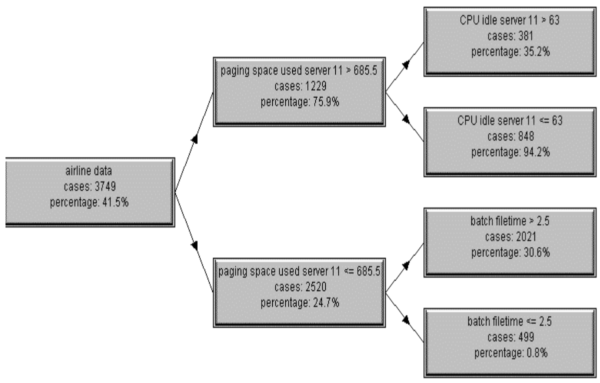

Figure 4: Decision tree

Top N

A top N was created for this dataset. We used information gain as ordering criterion. The resulting top 3 is displayed in Table 1.

Attribute	***Threshold***
Paging space used server 11 (MB)	> 685.5
Client 6 ping time (millisec.)	> 258.5
CPU Idle time server 5 (perc.)	< 74.5

Table 1: Top 3

The attribute *paging space used* is discussed in the previous section. The attribute *client ping time* is the ping time to a foreign router. It is clear that if this ping time exceeds 258.5 milliseconds the performance for foreign users is bad, but it also indicates that influences the local performance. It was explained by the system manager of the airline company that these high ping times correspond to the events that foreign users load a complete table from the database to their client. This table could be very big and the network connection to foreign countries have a narrow bandwidth. So this situation caused both the network as well as the application to be very busy for some time. Unfortunately, fixing this bottle-neck is very expensive: buy more bandwidth or redesign and reimplement a part of the application to reduce network traffic.

The presence of the attribute *CPU Idle time on server 5* is a direct indication that this machine is a bottle-neck for the performance of the application.

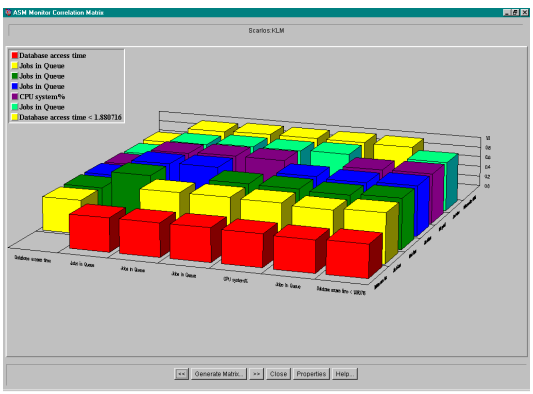

Figure 5: Association matrix

Associations

With the association rule algorithm we induced several types of rules. Simple rules that contain one monitor as condition and one monitor as conclusion are presented in an association matrix, see Figure 5. We found several surprising relations between individual components of the system. A striking example is the association of two disk IO monitors, the correlation between the disk IO of a disk in server 5 and a disk in server 25. After studying the data more closely it was concluded that these disks exhibited almost complete disjunct operation, see Figure 6.

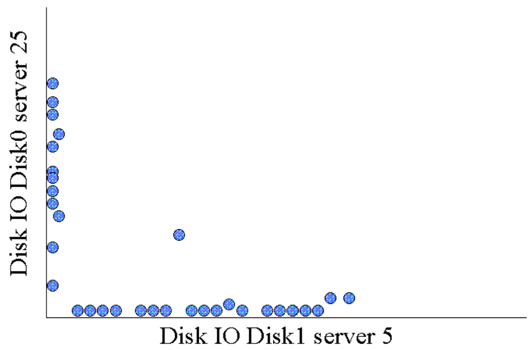

Figure 6: scatter plot of disjunct disks

Another use of the association rule algorithm is the induction of rules involving (multiple) monitors as condition and the SLA as conclusion. We found for example the following rules:

```
low freespace on /var/tmp on server 5
-> low performance

high paging space used on server 25 ∧
high # ftp connections on server 25
-> low performance
```

First order techniques

The next sections describe *ILP* experiments. To compensate for the growth of the search space by the addition of the model information and using a first order modeling technique, we discretized the data set into binary attributes incorporating only values that are more than 3 times the standard deviation above or below average. This reduces the data set size (leaving out all “normal” values) and simplifies the search (only binary attributes are used). The loss of information in this preprocessing step has been taken for granted because the main goal of applying first order techniques was to give a proof of concept of the usability of ILP. Using all values without reducing the search space may have produced better results but also would have dramatically increased the analysis time.

Progol

Progol was used to induce rules that incorporated the relevant model information.

Progol generated the two rules displayed in Figure 7.

```
Performance is low <-
 ∃ monitor(X, high) ∧
 monitorclass_id(X, #requests in queue)

Performance is low <-
 ∃ monitor(X, high) ∧
 monitorclass_id(X, NFS server).
```

Figure 7: progol rules

Because there is only one monitor of type *number of requests in queue*, this rule is essentially propositional. The queue contains the requests that still have to be dealt with. This means the application cannot handle all incoming requests. This could mean that too many users use the application at the same time. The second rule means that if any monitor of type *NFS server* on any server in the model is high the performance will be low in the coming period. Although there are just six monitors that have type *NFS server*, the second rule is very interesting because this gives a more general description of the problem, which is that on server 11, the network load and the usage of this server is too high. So what we already found in several propositional experiments (see decision tree) is expressed here in one understandable rule.

Tilde

Tilde was presented with the same discretized data as Progol. Tilde induced the tree in Figure 8.

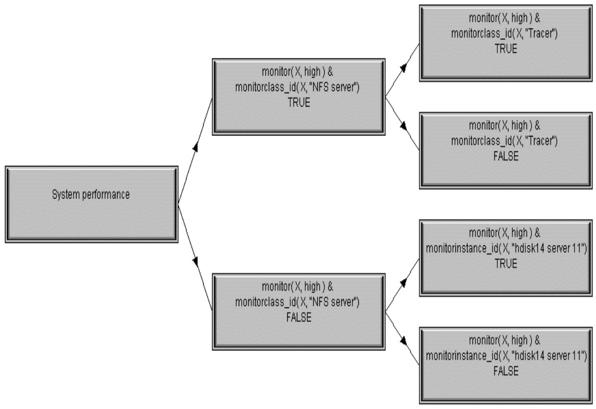

Figure 8: Tilde Decision Tree

The first split in the tree is equivalent to the following rule:

```
If there is a monitor with class NFS
Server that has value high then
performance is low.
```

This is exactly the same result as found with Progol. High monitors of the *NFS Server* class have the greatest impact on the performance. The next split is on high monitors of class *Trace*. These monitors are the different performance measures defined for this application. So, this split tells us that the different performance measures are similar in the case of high monitors of class *NFS server*. If there aren’t high monitors of class *NFS server* Tilde splits on high monitor values for monitor instance *Hdisk14 server 11*. A monitor instance is a collection of monitors on a specific component, so this comes down to the fact that high usage of disk 14 on server 11 is causing performance decrease. Further investigation shows that this holds for almost all disks of server 11. From the propositional experiments we concluded that memory problems or overloading of server 11 was the main problem of the system. Here we see something similar. When NFS activity (on server 11) is low, high disk activity on server 11 is the main bottleneck. It is fairly easy to identify this situation as “swapping”. The machine has a low CPU load, low network activity but is only swapping memory causing high disk activity. Again, the conclusion is that server 11 needs more memory or the number of applications on this server should be decreased.

Conclusions

The ideas and experiments presented in this paper demonstrate that the application of Data Mining techniques to the domain of Adaptive System Management may offer a lot of insight into the dynamics of the system at hand. In the experiments we found several unexpected and real problems with the system we analyzed. We have seen that, although propositional techniques are very useful, first order techniques could improve the usability of ASM considerably. Further work should be done in the area of an efficient implementation of first order techniques. Also further research should be done to determine domain specific properties that can be used to improve efficiency and quality of the results. A high priority of ASM is the understandability of the results, especially for System Managers, who are typically not Machine Learning experts. We gave a proof of concept for a large-scale implementation of ASM. Support effort can be reduced and systems can be analyzed on a regular basis to prevent critical systems and applications from crashing by an implementation of Adaptive System Management.

Acknowledgements

We wish to thank Stephen Muggleton & James Cussens of the University of York as well as Hendrick Blockeel of the University of Leuven for participating in and contributing to the ASM experiments performed at the University of York. Furthermore, we would like to thank our colleagues at Syllogic, especially Alex Bradley, Pieter Adriaans, Marc Gathier and Marc-Paul van der Hulst.

References

1. Adriaans, P.W. and Zantinge, R. *Data mining.* Addison-Wesley, 1996.

2. Adriaans, P.W. *Adaptive System Management*, in Advances in Applied Ergonomics, proceedings ICAE'96, Istanbul, 1996.

3. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A. 1996. *Fast Discovery of Association Rules*, in [4].

4. Blockeel, H., De Raedt, L. *Top-down induction of logical decision trees*, submitted to Artificial Intelligence, 1997.

5. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. *Advances in Knowledge Discovery and Data Mining,* AAAI Press/MIT Press, 1996.

6. Herlaar, L. *Diagnosing Performane of Relational Database Managament Systems*, technical report, Utrecht University, 1995.

7. Ibraheem, S., Kokar, M., Lewis, L. *Capturing a Qualitative Model of Network Performance and Predictive Behavior*, Journal of Network and System Management, vol 6, 2 ,1997.

8. Knobbe, A.J., Adriaans, P.W. *Analysing Binary Associations*, in Proceedings KDD '96.

9. Knobbe, A.J., Den Heijer, E., Waldron, R. *A Practical View on Data Mining,* in Proceedings PAKM '96.

10. Knobbe, A.J. *Data Mining for Adpative System Management*, in proceedings of PADD '97

11. Lavrač N., Džeroski S. *Inductive Logic Programming, techniques and applications,* Hellis Horwood, 1994.

12. Lavrač N., Džeroski S. *Inductive Logic Programming, proceedings ILP-97,* Springer, 1997.

13. Muggleton, S. *Inverse entailment and Progol.* New Generation Computing, 13:245-286, 1995.

14. Quinlan, J.R. *C4.5: Programs for Machine Learning*, Morgan Kaufman, 1992.

15. de Raedt, L. (Ed.) *Advances in Inductive Logic Programming,* IOS Press, 1996.

16. Zantinge, R. and Adriaans, P.W. *Managing Client/Server*. Addison-Wesley, 1996.