

Qualitative Performance Control in Supervised IT Infrastructures

Gergely János Paljak, Zoltán Égel, Dániel Tóth, Imre Kocsis, Tamás Kovács házy, András Pataricza

Department of Measurement and Information Systems

Budapest University of Technology and Economics

Budapest, Hungary

{paljak, egelz, dtoth, ikocsis, khazy, pataric}@mit.bme.hu

Abstract – The datacenter performability control today still lacks approaches that integrate well into the system management solutions present in enterprise settings and that are compatible with the skill sets of system administrators. We propose a methodology for designing qualitative, state-based predictive performability controls that use instrumentation already present in the form of a system monitoring framework and that perform corrective actions at the (virtual) host level. At the same time, our solution offers a way for addressing the typical problem of overmonitoring of enterprise environments. Based on our empirical results, we identify the main systemic defects of current monitoring tools that hinder designing trustworthy fine-granular controls utilizing them.

Keywords – performance control, mRMR, IT system supervision, overmonitoring

I. ON SENSOR SELECTION IN SYSTEM MANAGEMENT

Service and system management in data center environments has to ensure the fulfillment of Quality of Service (QoS) and generic dependability constraints imposed on the services provided. In order to be able to predict and respond to deviations from the service-level nonfunctional requirements, data center operations typically employ service and system monitoring solutions that gather and process various performance and resource characteristics in a quasi-online fashion. In the context of performability management, the processed measurements are used to generate alerts when the QoS requirements have been or are to be compromised. Upon alerts, usually semi-automated actions are taken to remediate the problems present. From the theoretical point of view, this constitutes a hybrid system control scheme with a performance characteristics facet that is of continuous nature and a decidedly combinatorial diagnostic aspect.

The industrial application practice of contemporary monitoring frameworks as IBM Tivoli Monitoring [1] or Nagios [21] utilizes mostly ad-hoc approaches for designing alerting configurations and the corrective action selection logic. Operators typically maintain a system that is a composition of black-box components from the performance as well as dependability characteristics point of view; also lacking the knowledge and resources to perform full control and diagnosis planning workflows.

As a result, such frameworks are generally deployed so that every applicable and “potentially important” data collector plugin or agent is used – from the hardware and physical network level up to applications and services.

Thus, systems are heavily *overmonitored*: a sizeable set of metrics is constantly gathered that is heavily redundant with respect to the usually coarse-grained available corrective actions. Additionally, the alerting and action rules are usually based on rules of thumb and informal knowledge about the system. In effect, trustworthy system management would necessitate much more well-founded online regulation and control approaches, as also identified by the Autonomic Computing initiative [4].

While applied classic control theory has already found its way into the performance management of IT systems (as demonstrated in the Related Work section), most of these approaches share the common property that they apply the workflows of classic control theory premising that the set of variables upon which the plant model will be defined is known. This can even be a valid assumption for the the black-box modeling of well-defined solitary components, platforms with well known behavior, or the analytical modeling of white-box components. Also, compositional modeling of heterogeneous distributed systems with black box components may assume the “important” variables to be known. However, when the whole system acts as one component with a large number of internal variables from the performability point of view – as it is all too realistic for general purpose IT systems – such assumptions are simply not met.

For these cases, there is a currently unaddressed prerequisite for control design. While the variables to be controlled – QoS – and the applicable “actuators” – in the era of virtualization and cloud computing, more and more simply scaling the number of servers used – are largely known beforehand, the internal state variables of the system that are of interest are not. From the massive amount of properties that are measurable (and are typically indeed measured), a subset must be chosen that is appropriate for control design. This requirement is generic; it is present for simple rule-based as well as true closed loop control. Hence, we call this initial step the *sensor selection problem* of IT management. Solving the sensor selection problem has the added benefit of discarding those variables from the set of all measurable metrics that are

redundant from the point of view of the objective of the control. As such, it addresses the problem of overmonitoring at the same time.

Nevertheless, the connection between the application domains of classic control problems and IT service management runs much deeper than the need for an “autonomic control loop” and borrowing selected techniques. Modern applied control theory addresses systems that are reconfigurable, on the plant as well as the controller level; the architectural pattern (see Figure 1.) [17] necessary for such controls is readily applicable for IT service performability management in data centers.

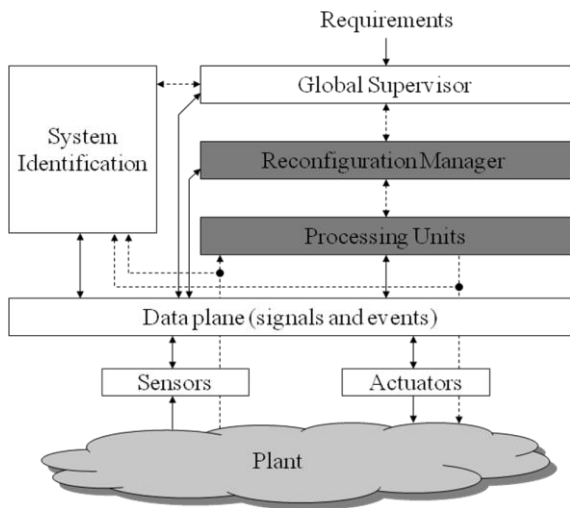


Figure 1. Reconfigurable control architecture for classical control systems

We argue that utilizing the results of classical control in IT service management in the practice today is hindered by mostly three factors: a) the problem of *sensor selection*, b) the lack of lightweight and properly supported *model and system identification* and c) the *inadequacy of the available sensors* from a measurement theoretical point of view.

This paper mainly proposes a novel solution for the first topic, while addressing the second with a generic approach that is adequate for a wide range of systems. We offer some insight on the third topic in our concluding remarks.

II. PERFORMANCE CONTROL WITH OFF THE SHELF COMPONENTS AND SUPERVISION

In this paper we introduce an approach for designing proactive rule-based performance control for multi-tiered, distributed On-Line Transaction Processing (OLTP) systems, where control can influence the system by adding and removing servers to and from tiers.

We model the system-wide internal performance state in a high-level, qualitative way. Our previous results [18] have shown that at any moment, the system can be in one of the following operational domains: NORMAL, DEGRADING, SATURATED.

The single most important advantage of such highly qualitative, state-based models is that it establishes a familiar

diagnostic interface for system engineers, based on which they can easily state the necessary control actions as rules. For example, a business requirement induced rule can be that if a high priority service switches (or will switch) to the “DEGRADING” state from “NORMAL”, then a server should be reallocated to it from a lower-priority service that is (or will be) in the “NORMAL” state. In terms of classical control theory our system model is practically a piece-wise model of the plant, or using different terminology a system with modes of operation.

We assume that a monitoring framework is present in the system that has a very wide selection of measurable system and service metrics. We will perform sensor selection on this set in order to utilize the selected variables for *predicting the operational domain of the system on-line*, as defined by the simple qualitative model. The classification is used by state-based reconfiguration rules to select the action to be taken that is necessitated by the business-level resource allocation logic.

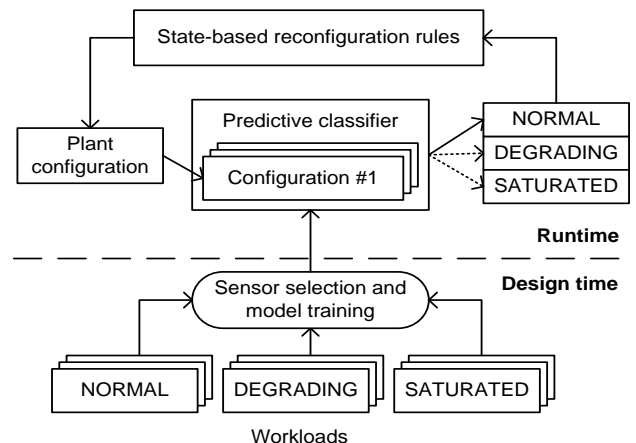


Figure 2. Operational domain prediction and control

At design time, sensor selection and the training of the predictive classifier is performed with a number of representative workloads and their transitions that are known to bring the system into the respective states (see Figure 2.). Note that while from the training point of view, we effectively teach the predictor to discriminate between specific workloads, these workload classes have associated fairly distinct steady-state QoS intervals. As such, the end effect is that the model associates internal variables and their dynamics to QoS classes that have been defined by the workload classes. Also, the model will include an amount of class transition characteristics.

This notion of operational domains and way of exciting a system for identification is arguably suboptimal from the control theoretical point of view, the most critique being appropriate regarding excitation frequency and signal shape. However, for typical IT systems there are some practical constraints that can be overcome only with relatively big efforts and that inevitably lead to this approach. (i) The monitoring frequency that is usually allowed by monitoring frameworks lies in the tens of seconds range (at the least). (ii) Even if the frameworks allow more frequent monitoring after fine-tuning, the software instrumentation itself may be too resource intensive or simply impossible to run at a greater

frequency. (This is mostly true at the application framework and server level.) (iii) The enterprise-grade tooling available for the performance testing of services (see e.g. IBM Rational Performance Tester) typically allows recording transactions and then running them with varying request parameters, request intensity, user data, number of users, and so on. Their goal is to support testing for specific cases (workloads parameterized by hand), not to excite the system in the most complete way possible.

Lastly, it is to note that service performability control by adding and removing (virtual) machines has an inevitable action delay that is big enough for such controls not be able to handle very rapid workload changes. Coupled with the fact that OLTP systems have a very limited memory of their historical state on such time scales, our approach is a practically feasible one despite its theoretical shortcomings.

III. RELATED WORK

The application of classic control theory to regulate the service-level performance of IT systems is a well more than a decade old field. Here we recall only some of the more significant examples in support of the arguments underlying our approach.

[1] applies control theory to web server QoS and presents a middleware to augment generic software services with control. [2] presents an adaptive optimal MIMO control for the request scheduling of services sharing resources in order to optimize the QoS of all services. [3] controls QoS of services by resource entitlement adaptation in resource-partitioned and virtualized environments. Recent versions of the IBM DB2 database apply control theory to e.g. memory tuning and database utility throttling [5]. Authors of [16] use control models to solve hot spot contention in network-level congestion control. In [15], a control is designed to meet delay requirements while saving energy by adaptively scaling frequency/voltage in a simulated portable device.

Larger-scale, cluster- or datacenter-level, control approaches also exist. Authors of [13] suggest using proportional thresholding to control the number of virtual machines allocated to service. The fact that only CPU utilization is used as input simplifies monitoring, but leaves the question of handling non-CPU constrained situations open. In [14] a cluster is controlled to achieve SLA requirements of a provided service while optimizing energy consumption. The authors use a limited lookahead controller (LLC) to drive the dynamic provisioning of services in a virtualized environment.

More in-line with the approach presented in this paper, previous research has already explored statistical and probabilistic solutions for forecasting the performability status of IT systems. [9] presents a best practice framework for resource forecasting in computer systems that includes a breakdown of the typical tasks encountered.

Probably the most difficult problem in statistically processing monitoring data is identifying and quantifying non-linear relations among time series. [11] uses statistical learning in the form of so-called Tree-augmented Bayesian Networks (TANs) to identify metrics and thresholds of system

monitoring that correlate to such high-level properties as whether the system complies with Service Level Objectives (SLOs). The core of the approach implements a greedy feature selection over TANs, and focuses on running diagnostics on historic data. Predictions and on-line model modification are not provided, however.

In [12] the non-linear interdependencies among metrics are quantified by mutual information that in turn is used as a measure of similarity. Similar metrics are clustered and intra-cluster entropy is calculated to track the health of enterprise systems. Anomaly detection is performed by recognizing significant shifts in entropy.

IV. FEATURE SELECTION AND PREDICTION

The sensor selection problem is essentially a task of dimension reduction. Dimension reduction is the process of reducing the number of considered variables of the mathematical model of a problem. It is a well-known topic in statistical and machine-learning, where datasets of high dimensionality are commonplace.

Dimension reduction methods are generally divided into feature selection and feature extraction approaches. While feature selection chooses a subset of the original dimensions, feature extraction projects the original problem space into one of fewer dimensions. [7][8]

In our case, we aim at selecting subsets of the measurable metrics that “adequately describe” the behavior of the system in various operational domains for prediction purposes. The underlying rationale is that for a given service for a given load class, we will inevitably confront “clusters” of variables whose time series are similar by shape, and thus are redundant for ad-hoc system management as well as prediction.

As in our previous work [18], we use the bioinformatics inspired minimum Redundancy maximum Relevancy (mRMR) algorithm [19] for feature selection. Although greedy by nature, mRMR selects an optimal set of variables instead of individual “best” ones, thus it aims at covering every defining aspect of the examined system with a minimal set of features.

mRMR is based on the concept of mutual information, defined for two probabilistic variables x , y as:

$$I(x; y) = \iint p(x, y) \log \frac{p(x, y)}{p(x)p(y)} dx dy$$

We want to select a set of variables, S so that the mutual information between each element of S and the objective metric C is maximal (maximum relevance):

$$\max R(S, c), \quad R = \frac{1}{|S|} \sum_{x_i \in S} I(x_i; c)$$

and the redundancy is minimal inside S , which means that the mutual information between the elements is minimal:

$$\min r(S), r = \frac{1}{|S|^2} \sum_{x_i, x_j \in S} I(x_i; x_j)$$

We intend to find those attributes that have the highest mutual information against an objective metric, and keep the mutual information low among the set of the identified attributes in order to find signs of distinct performance issues. In practice an iterative algorithm optimizes the following condition:

$$\max_{x_j \in X - S_{m-1}} \left[I(x_j; c) - \frac{1}{m-1} \sum_{x_i \in S_{m-1}} I(x_j; x_i) \right]$$

where $m = |S|$, the size of S in the current iteration and S_{m-1} being the set of metrics selected prior to the current iteration.

It is to note that mRMR by itself does not take the timeseries nature of our measurements into account (mutual information being defined over probabilistic variables). However, by the finite ‘‘unfolding’’ of the data in time (representing variable value at earlier points of time as separate variables) the discovery of delayed effects can be supported.

While other approaches exist for feature selection (see the Related Work section), we have chosen mRMR due to its nature for searching for variables that ‘‘complement each other’’. While an in-depth comparison is out of the scope of this paper, our application experiences during the work presented here, the one reported on in [18] and others conducted during an industrial pilot project clearly show that mRMR is almost always a better choice for variable selection in this setting than the classic, linear approaches, while in contrast to feature extraction methods, maintains the original meaning of variables.

We employ neural networks as predictive classifiers. Neural networks are not only universal approximators, but also have the capability to adapt with constant learning to minor changes in the system that is predicted.

Again, while there could be some other candidates for realizing predictive classification apart from neural networks, our experience shows that in the systems we have examined linear methods (in particular, linear regression) deliver unacceptable results.

V. EXPERIMENTAL SETUP AND RESULTS

Our pilot application infrastructure emulates a scaled-down datacenter. A monitoring and control framework is deployed that measures and processes software and platform performance metrics (Figure 3.).

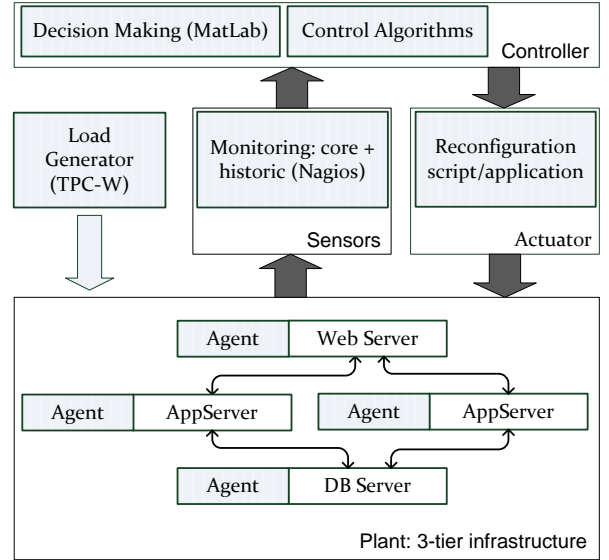


Figure 3. Experimental setup

The TPC-W benchmark [22] is deployed on the infrastructure as a widely accepted benchmark for OLTP systems. We use a Java Servlet based implementation, described in [23], to drive the experiments. The workload is generated according to the TPC-W specification by using emulated browsers.

The OLTP infrastructure and the monitoring and control framework are installed on virtual machines that are deployed on two separate physical machines. This setup avoids measurement bias introduced by supervisory machines with physical separation. Note that resource contention and hypervisor overhead on the server hosting the OLTP components would be significant factors if we would aim at compositional modeling [6]. However, our system-level quasi black box approach is applicable for such environments.

The infrastructure is composed of a database backend (MySQL server), two application servers (Apache Tomcat) with the TPC-W application installed, and a web frontend (Apache httpd). The application servers are organized in a load-balanced, fault-tolerant cluster; the number of application servers can be dynamically changed in runtime.

Every infrastructure node is equipped with a monitoring agent of Nagios, a widely-used open source system monitoring solution. The platform- and application-level performance data is measured every ten seconds by the agents and collected in a central data store. The data store is queried by the decision making unit (Matlab) for analysis and model building (off-line) and for model evaluation, decision making and actuation (on-line). Actuation – adjusting the number of application servers – is carried out in real time, through a simple Java application written for this purpose.

We use a workload profile that stresses the infrastructure in all three operational domains. A single experiment takes two hours and is composed of 5 phases of equal durations: it starts in the ‘linear’ domain (20 parallel users), then steps to the ‘degrading’ domain (50 parallel users), then steps to the ‘saturated’ domain (70 parallel users). After saturation it drops

to ‘degrading’, and finally to ‘linear’. We measure the available system-internal metrics and throughput as a QoS metric. This experiment is repeated to provide our training and validation datasets. We use 75% of our experiments for feature selection and to train the neural networks and 25% to validate the approach; results shown here are from 8 single experiments (16 hours). During load generation we follow the TPC-W standard; the specification-prescribed random factors lead to a very high variance (Figure 4.).

mRMR feature selection is performed on the measured variables, their three cycle deep temporal unfolding and the future value of the QoS metric at the temporal horizon we aim to predict – in our experiments, 1, 3 and 5 minutes, respectively. We select metric sets with 8 members – currently, the number of metrics to be selected is determined in an ad-hoc way.

It is to note that the selected feature sets all include the actual and unfolded QoS metrics; more interestingly, the CPU utilization of the Apache web server and one Tomcat server are also present in all three cases. This not only conjectures that our system is mostly CPU-intensive, but also that CPU usage dynamics are not trivial – otherwise the CPU metrics would have been put into one implicit cluster by mRMR, with one single representative being selected.

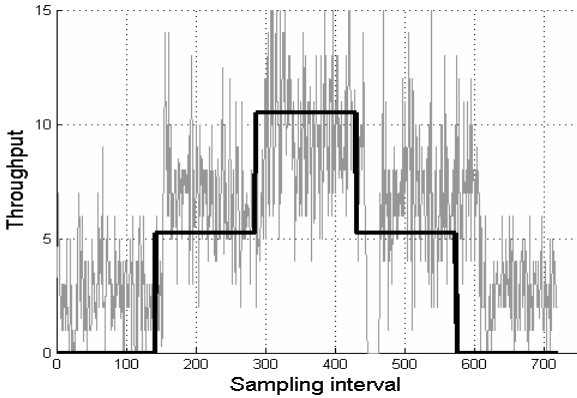


Figure 4. Sample teaching load; operational domains are superimposed

As a next step, for the three prediction horizons we train three neural networks with a QoS value based state classification resulting from the teaching runs (more precisely, we apply feed-forward neural networks with a single hidden layer). The resulting networks are simulated with measurement data separated for validation. The output is post-processed, mapped to the discrete values defining the operational domain and fed to a simple decision algorithm. The decision algorithm only allows a change of operational domain prediction if outputs are equal in a certain window length aimed at reducing the stochastic noise of TPC-W, avoiding an oscillation of state changes. Additionally, we have also implemented a ‘safe-mode’ prediction. In this case there are two windows: one for changing prediction upwards (shorter windows), and one for changing downwards (longer window). This is justified by the fact that overusing resources (de-provisioning too late) is generally more acceptable than breaching SLAs (de-provisioning too early).

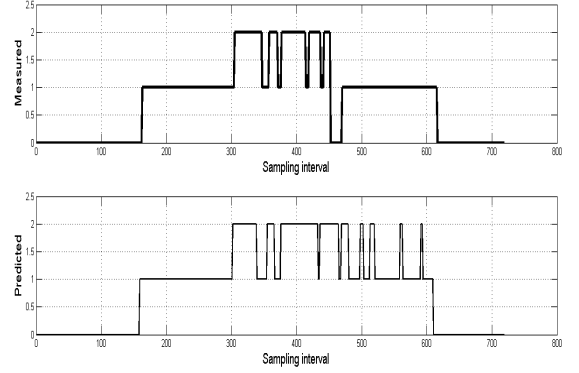


Figure 5. Measured QoS class (top) and Predicted QoS class (bottom)

Results show 84-86% prediction accuracy (an example is shown in Figure 5.) in operational domain recognition of all three time horizons, with a large proportion of inaccurate predictions originating from the degrading state, which again could be avoided by implementing a more sophisticated decision algorithm.

The above introduced results differ from those shown in our previous work [18] for one practical reason, namely the lack of additional database back-ends and their supporting middleware. As a crucial effect, the stochasticity of TPC-W are not scattered on several back-ends, but influence directly the single underlying instance. As a result, not only do the measured system internal metrics show a greater variance due to the lower number of components, but also some undocumented flaws in the non-deterministic of the TPC-W implementation cause time and again unwanted discrepancies, thus making the distinction between statistical fluctuation and actual state-changes extremely challenging, as Figure 6. demonstrates.

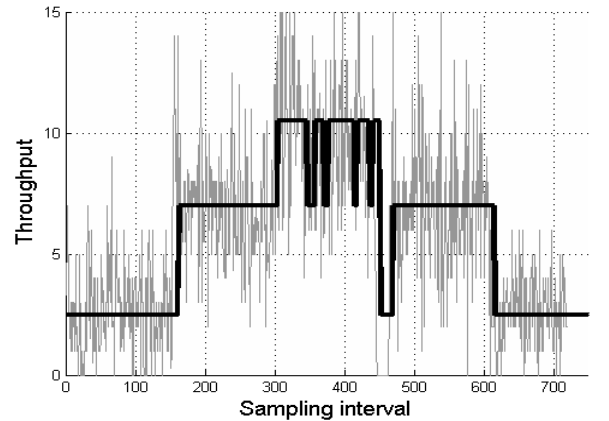


Figure 6. A TPC-W load with QoS value based classification superimposed

VI. CONCLUSIONS

The most important conclusions we have drawn during our work concern the widely available monitoring frameworks and

the way resource allowances can be managed at the (virtual) server granularity.

From the point of view of performability control, current monitoring frameworks (commercial as well as open source) and software/hardware instrumentation for monitoring share some systematic faults. On the one hand, measurements of various metrics are typically not synchronized; thus, even when the timestamp of a measurement is accurate (what is by far not always the case), at least interpolations have to be performed on the gathered data to artificially synchronize the time series, thus losing signal fidelity.

On the other hand, the minimum of the measurement intervals that is typically allowed by the frameworks is so big, that it makes applying classic feedback control theory for performance management infeasible without custom monitoring instrumentation. On a more practical level, while filtering out high-frequency metric behavior components may be acceptable for simply diagnosing the current state of the system from monitoring cycle to monitoring cycle, it can seriously impact the achievable performance of prediction.

Another aspect of the problem is that the monitoring instrumentation of many software components is so slow and resource intensive, that polling is not only infeasible in the some hundreds of milliseconds, but even in the tens of seconds range.

To address the first two points, we have begun exploring the design necessary for a monitoring framework that is amenable towards true closed-loop performance control. We believe that a solution is needed where measurements are synchronized by a global clock, the results are stored locally on the nodes and are served upon (remote) request. Alas, inefficient software instrumentation is a much harder problem. While it theoretically can be addressed by case-by-case modifications at least for open source systems, this is clearly not a solution that fits our stated goals. Instead, we suspect that many of such “heavyweight” metrics are in close correlation with other, more “lightweight” ones. Thus, sensor selection should be able to “trade” metrics with inefficient instrumentation for a small set of metrics that can be efficiently gathered.

Currently, adjusting resource allowance at the virtual server level as a corrective action has some serious limits, too. On typical cloud computing platforms, we can only cold boot and shutdown machines, what results in a significant action delay. While in-house virtualization (e.g. VMWare ESX) adds the possibilities of using snapshots and partitioning physical resources, virtual machine resource allocations cannot be modified on the fly for powered up machines – with the significant exception of “Dynamic Logical Partitioning” in IBM PowerVM Virtualization. Consequently, transparent fine-granular performance control in the most widely used virtualized environments still lacks some of the instrumentation it would necessitate.

REFERENCES

- [1] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in software services," *IEEE Control Systems Magazine*, vol. 23, 2003, p. 74–90.
- [2] M. Karlsson and C. Karamanolis, "An Adaptive Optimal Controller for Non-Intrusive Performance Differentiation in Computing Services," *2005 International Conference on Control and Automation, IEEE*, 2005, pp. 709-714
- [3] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control of resource containers on shared servers," *9th IEEE Intl Symposium on Integrated Network Management, IEEE*, 2005, pp. 163-176..
- [4] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, 2003, pp. 41-50.
- [5] S.S. Lightstone, M. Surendra, Y. Diao, S. Parekh, J.L. Hellerstein, K. Rose, A.J. Storm, and C. Garcia-Arellano, "Control Theory: a Foundational Technique for Self Managing Databases," *Proc. 2nd Intl. Workshop on Self-Managing Database Systems*, 2007, pp. 395-403.
- [6] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, 2010, p. 55.
- [7] Fodor, I. K. "A survey of dimension reduction techniques". Technical Report UCRL-ID-148494, Lawrence Livermore National Laboratory, Center for Applied Scientific Computing, 2002.
- [8] L. Molina, L. Belanche, and A. Nebot. "Feature selection algorithms: a survey and experimental evaluation." *International conference on data mining*, Maebashi City, Japan, 2002.
- [9] G.A. Hoffmann, K.S. Trivedi, and M. Malek, "A Best Practice Guide to Resource Forecasting for Computing Systems," *IEEE Transactions on Reliability*, vol. 56, 2007, pp. 615-628.
- [10] R. Kohavi and G. John. "The wrapper approach". In H. Liu and H. Motoda, editors, *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Springer Verlag, 1998.
- [11] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J.S. Chase, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," *In Proc. 6th USENIX OSDI*, 2004.
- [12] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. S. Ward. "Information-theoretic modeling for tracking the health of complex software systems". *In Proc. 2008 Conference of the Center for Advanced Studies on Collaborative Research.*, ACM New York, 2008.
- [13] H. Lim, S. Babu, J. Chase, and S. Parekh, "Automated control in cloud computing: challenges and opportunities," *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ACM New York, NY, USA, 2009, p. 13–18.
- [14] D. Kusic, J.O. Kephart, J.E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, 2008, pp. 1-15.
- [15] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron, "Control-theoretic dynamic frequency and voltage scaling for multimedia workloads," *In Proc. 2002 Intl. Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2002, p. 156–163.
- [16] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving hot spot contention using infiniband architecture congestion control," *Proceedings HP-IPC 2005*, 2005, pp. 41-44.
- [17] G Karsai, G Biswas, S Narasimhan, T Szemethy, G Peceli, G Simon, T Kovacszhazy, "Towards Fault-Adaptive Control of Complex Dynamic Systems," *In: Software-Enabled Control: Information Technologies for Dynamical Systems*. IEEE Press, 2003, pp. 347-368.
- [18] G. Paljak, I. Kocsis, Z. Egel, D. Toth, A. Pataricza, "Sensor Selection for IT Infrastructure Monitoring," *In: Autonomic Computing and Communications Systems*, Springer, 2010, pp. 130-143.
- [19] H. Peng, F. Long, C. Ding: "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 1226-1238, 2005.
- [20] IBM Tivoli Monitoring product page. <http://www-01.ibm.com/software/tivoli/products/monitor/>
- [21] Nagios home page. <http://www.nagios.org/>
- [22] TPC-W official page: <http://www.tpc.org/tpcw/default.asp>
- [23] TPC-W Java implementation: <http://www.ece.wisc.edu/~pharm/tpcw.shtml>

DRAFT