Framework for Consolidated Workload Adaptive Management

Marcin Jarząb[1] , Krzysztof Zieliński[1]

[1] Institute of Computer Science, AGH - University of Science and Technology,
Al. Mickiewicza 30, 30-059 Krakow, Poland
{mj, kz}@agh.edu.pl
http://www.ics.agh.edu.pl

Abstract. This paper presents the applicability of modern virtualization technology for policy-driven adaptive workload management. The workload consolidation problem and the resource management process are defined. Virtualization technologies used for workload consolidation haven been shortly overviewed and compared. Particular attention is devoted to lightweight virtualization using a container technology, applied in the Solaris 10 Operating System. Subsequently, software architectures for adaptive control of virtualized resources are described. Open-loop systems which exploit the model of controlled system behavior are compared to closed-loop systems exploiting feedback control. Practical aspects of the implementation of such systems for Solaris 10 containers have been elaborated. Construction of a controller for work consolidation is elaborated. The JMX technology and the Solaris 10 container technology are used for this purpose. Various principles of controller implementation have been proposed and practically verified. The conditions of their applicability have also been identified. As a conclusion, a hybrid controller has been constructed and successfully applied in an experimental system.

Keywords: software engineering, virtualization, adaptive, workload management

1 Introduction

Efficient exploitation of complex computer systems under changing workload conditions requires greater IT infrastructure flexibility through intelligent matching of computing resources to application requirements, in order to meet Service Level Agreements (SLA). This may be achieved by equipping computer systems with mechanisms supporting self-management – a common trend observed recently in initiatives promoting the vision of Autonomic/Adaptive Computing [1]. They aim to build computing architectures that are capable of managing themselves, anticipate workloads, optimize performance and adapt to events occurring in the surrounding environment. In this way they contribute to QoS application requirements, which are of increasing importance in modern software engineering.

Autonomic computing introduces the Autonomic Manager concept which performs the following activities: monitoring, analyzing, planning and execution. These processing steps require knowledge about managed resources built into the manager or collected during runtime. In the former case, such information can be represented as mathematical formulae, i.e. a managed resource model describing a relation between its state and control action that should be performed to achieve desired system behavior. The latter case corresponds to a situation where such a model is difficult to identify. The manager may search for suitable control parameters during an iteration process performed directly over managed resources. The managed resource has to be equipped with an interface providing sensors and effectors functionality to be managed.

In order to achieve manageability, the computing resources are virtualized. *Virtualization* is a technology which combines or divides computing resources to create one or many operating environments, using such methodologies as hardware and software partitioning or aggregation, partial or complete machine simulation, emulation, timesharing, and many others. As virtualization provides mechanisms for control of resources the complexity of the computer system remains very high and setting up a suitable control algorithm is challenging. The efficiency and correctness of the control strategy depends on many parameters, all of which must be very carefully identified.

The research presented in this paper exploits lightweight containers as a virtualization technology used for load consolidation in modern operating systems. The aim of this study is to investigate the software architecture framework for consolidated workload management built with support of the JMX [7] technology and a preliminary study of different control strategies. Particular attention is devoted to identification of conditions under which classical control theory can be applied to control CPU-bound workload. Practical experiments have been performed for Solaris 10 containers.

The structure of this paper is as follows. In Section 2, workload consolidation mechanisms provided by modern virtualization and management technologies are discussed. The resource management problem is defined in this context. Subsequently, in Section 3, the concept of software architectures for adaptive control of virtualized resources are described. The problem of exposing virtualized operating system resources as managed resources, instrumented using sensors and effectors, is also discussed. Section 4 presents practical aspects of the implementation of such a workload management system. Open-loop systems which exploit the model of managed system behavior are compared to closed-loop systems exploiting feedback control. Construction of regulators for work consolidation management is elaborated. The JMX technology and Solaris Container technology are used for this purpose. In Section 5, different principles of controller implementation have been proposed and experimentally verified. The conditions of their applicability have been identified. The paper ends with conclusions.

2 Virtualization technology for workload consolidation

Modern enterprise data centers are shifting towards a utility computing model where many business-critical applications share a common pool of infrastructure resources that offer capacity on demand. Management of such a pool requires having a control system that can dynamically allocate resources to applications in real time. Each physical machine in the pool can consist of a number of virtual containers, each of which can host one or more applications.

Enterprise applications typically have resource demands that vary over time due to changes in business conditions and user demands. This poses new challenges for system and application management which do not exist in dedicated environments. Because each of the hosted applications can express a resource demand that changes in short time scales (e.g. seconds or minutes), there has to be a control system that can dynamically allocate the server's capacity to virtual containers in real time. The benefit of doing so is that it allows statistical multiplexing between resource demands from co-hosted applications, so that shared servers can reach higher resource utilization. At the same time, the control system should be responsive enough to ensure that application service level objectives (SLOs) can be met.

The big challenge for consolidating multiple applications into a single physical server is to provide mechanisms of control over the resources (e.g. CPU, memory portions or network bandwidth) utilized by those applications. In the case where consolidated applications must be grouped (e.g. by business importance) into hierarchical sets, called *workloads*, simple tools are not sufficient.

Modern, advanced operating system environments provide mechanisms to better satisfy performance requirements of workloads called *lightweight virtualization*. There are two primary approaches to this virtualization:

- *Container-based*, which involves software that virtualizes an operating system environment. There is only one underlying operating system kernel, which the containers enhance by providing distinct borders offering increased isolation between groups of processes. Containers do not emulate any of the underlying hardware. Instead, the virtualized OS or application communicates with the host OS to share resource usage, which then makes the appropriate calls to real hardware. This technology is explored for instance by OpenVZ [9], and Solaris Containers [5, 6].
- *Paravirtualization* virtualizes parts of an operating system environment but also selectively emulates the hardware devices that a virtualized OS requires. Paravirtualization provides both a virtual machine and access to the native hardware, and thereby lets users run many instances of different OS's. The best known examples of this concept are VMware [8] and Xen [10].

In the presented study the first kind of lightweight virtualization technology will be explored, as used in the Solaris 10 operating system. Solaris 10 supports even lighter virtualization mechanisms than Containers, offering resource control called *Projects*. A Project serves as an administrative tag used to group related work in a manner deemed useful by resource management without providing the isolation level supported by containers. Solaris Containers deliver predictable levels of quality of service. Some of these are achieved due to scheduling, which is a resource sharing mechanism that refers to making a sequence of resource allocation decisions at specific intervals. An application that does not need its current allocation leaves or is detached from the resource, which is then made available for another application's use.

When considering CPU resource consumption on a single machine by multiple workloads or isolated domains, e.g. Solaris Containers (which also contain several workloads), one must consider a situation where one workload or an entire domain monopolizes available processor cycles and impacts others workloads or domains. The default Solaris thread scheduler configuration is called Time Shared Scheduling (TSS). TSS adjusts the priority of each thread based on the time a given thread consumes or spends waiting for CPU resource and time quantum which is the limit of time for which a thread is assigned access to CPU, depending on priority. This might lead to a situation where important workloads suffer from insufficient CPU time to complete their work. It is desirable to have a scheduler which gives the ability to prioritize access to CPU resources based on the importance of the workload.

The concept of a Fair Share Scheduler (FSS) [13] was introduced by J. Kay and P. Lauder [12]. This scheduler provides precise control of CPU use, allowing optimal system CPU resource usage. The system administrator expresses the importance of each workload by the number of shares, which are not the same as CPU percentages: shares define the relative importance of a given active workload to other active workloads. If (i) S_w – shares assigned to workload W, (ii) N – number of active workloads, (iii) S_i – shares assigned to active workload i={1,..,N}, then the relative entitlement E_w of workload W can be expressed with the following equation:

$$E_w = S_w/\sum_i^N S_i \qquad (1)$$

It is important to emphasize that FSS only limits the CPU usage if there is competition for CPU; otherwise CPU shares are never wasted and a given active workload can use 100% of CPU resources. A thread in the FSS class is assigned CPU access by the scheduler according to its share allocation, recent utilization and CPU usage by the other threads. When considering a workload with multiple threads, the processor cycles are assigned proportionally between these threads according to shares assigned to the workload.

In Solaris 10, resource controls *project.cpu-shares* and *zone.cpu-shares* are used to specify CPU shares for Projects and Containers respectively. This feature is called *Two-Level Fair Share Scheduling*. *Zone* entitlement for CPU shares is also partitioned between *projects* in the *zone*. The values of these resource controls can be set statically or changed dynamically during runtime.

3 Software architecture for adaptive workload management

A very well known architecture for adaptive or autonomic computing has been proposed by IBM and distributed as the PMAC [2] (Policy Management Autonomic Computing) toolkit. PMAC uses the very general concept of AE (Autonomic Element) depicted in Fig.1. It consists of an AM (Autonomic Manager) and MR (Managed Resources). The control loop of AM consists of four basic steps:

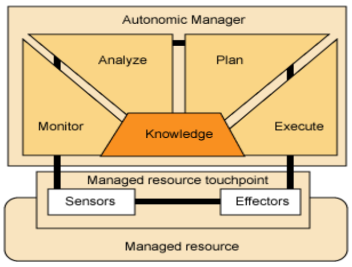

Figure 1.Autonomic Manager architecture proposed by IBM [1].

monitoring, analyzing, planning, and executing, which are typical for adaptive systems. These steps can exploit knowledge collected during system activity or supplied in the form of some kind of model. MR represents computer resources instrumented with sensors and effectors. Usage of the PMAC AM model for adaptive workload management is quite natural. It requires exposition of virtualized OS components such as Solaris OS Containers as MR, and implementation of AM. The most convenient way is to implement sensors and effectors for Solaris Containers using JMX technology and present them to a decision subsystem as Java managed beans (MBeans). The solution is part of the JIMS [4] platform, developed by the authors, described in more detail in [3] and roughly illustrated in Fig. 2. It's designed as a JIMS extension module implemented as a set of MBeans. Each container and project have separate instances of *Effector* and *Sensor* MBeans. The MBeans wrap the native OS mechanisms of project and container resource management and expose them as JMX connectors. Connectors make it possible to couple MBeans with the decision subsystem using any of the available RMI, HTTP or SNMP communication technologies.

In Solaris 10 consumption of resources for workloads can be measured using the *prstat* command. This command reads data stored in */proc* VFS (Virtual File System), where each process in the zone has its own subdirectory (in the global zone there are also entries for processes from local zones). The most important information is contained in *psinfo*, *usage* and *status* files.

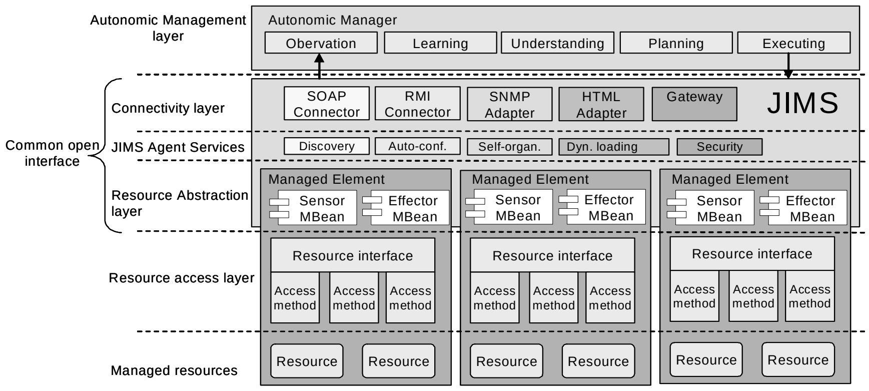

Figure 2. JIMS extension modules for Solaris 10 management.

By summarizing information about resource usage of processes it becomes possible to calculate resource usage of *projects* and *zones*. The capability to review historical data is provided by the *Extended Accounting* facility which allows collection of statistics at the process level, task level or both. Referring to these OS mechanisms, MBeans are divided into three groups: monitoring, management and accounting. Inside each group there are MBeans for *zones* and *projects*. Resource monitoring MBeans contain read-only attributes with basic information about zones or projects and their resource usage (CPU, memory, threads etc.) This information is periodically retrieved from */proc* VFS with the help of a native library accessed through Java Native Interface (JNI). Management (Effector) MBeans use various methods to interact with the OS: in order to collect information about *zones* and *projects,* MBeans read configuration files or use JNI. Changes in configuration are applied by executing shell scripts and system commands (via invoking Java `Runtime.exec()`). MBeans are also able to emit JMX notifications to inform interested parties about changes in the system (i.e. concerning added projects or changed resource usage).

Implementation of an Autonomic Manager (AM) with the PMAC toolkit is not straightforward due to some drawbacks. The current implementation, when used in a distributed environment, involves the Webshpere Application Server, while remote interfaces are only accessible via *Enterprise Java Beans* components. There is no support for JMX, which is a technology widely used for software system management [4]. Moreover, only *Common Base Events* can be consumed by PMAC, which calls for suitable adapter construction.

4 Practical aspects of workload controller implementation

Taking into account the drawbacks of the PMAC toolkit, a decision was made to implement AM using the JMX technology based on the JIMS framework. If a given event occurs (e.g. if there is a load change or a resource monitoring notification emerges) it's possible to react more effectively when consumers are registered directly within the AM. Furthermore, even though having several control loops in system coordination seems desirable, the implementation of such behavior seems to be a non-trivial task with the PMAC. Thus, the customized AM is geared for workload management of Solaris Containers based on mechanisms specified by the *Control Theory* [11] and structured as:

- Open-loop AM workload manager, exploiting the FSS model. The number of active Containers or Projects and their share (S_i) assignment is monitored. Equation 1 from Section 2 is used for relative entitlement E_w of workload W calculation and suitable S_i adjustment.
- Closed-loop AM workload manager, which directly tunes Containers' or Projects' resource shares to achieve desired CPU allocation to the workload. In this case, the CPU consumption U_w^t at time t is measured and used by the controller to calculate desired shares. The key aspect of such an AM is the controller algorithm concept.

The proposed AM types are depicted in Fig.3. Both managers use the same resource allocation control mechanisms, provided by FSS, but they differ in share calculation procedure. Examples of such procedures are considered below.

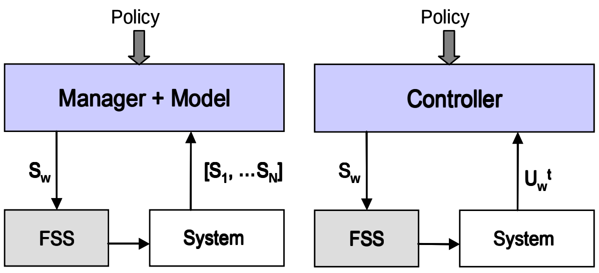

Figure 3. Open-loop and closed-loop AM concept.

4.1 Closed-loop control using Proportional or Proportional-Integral regulators

The model is not used in this case; the whole system is treated as a black-box using the closed-loop controller depicted in Fig. 3. The controller uses current CPU usage values and adjusts them by changing shares (control signals) to maintain the requested CPU usage.

A sample controller algorithm could use the *Propotional (P)* regulator expressed by equation (2) and *Proportional-Integral (PI)* where: (i) U_w – required usage of CPU by workload W_w, (ii) U_w^t – usage observed at time t by workload W_w, (iii) S_w^t – number of shares set at time t for workload W_w, (iiii) K_p – proportional coefficient, K_i – integral coefficient.

$$\mathbf{S_w^{t+1}} = \mathbf{S_w^t} + K_p * \mathbf{e(t)}, \text{ where } e(t) = U_w^t – U_w \qquad (2)$$

$$\mathbf{S_w^{t+1}} = \mathbf{S_w^t} + K_p * \mathbf{e(t)} + K_i \sum_i^t \mathbf{e(t)} \qquad (3)$$

4.2 Open-loop regulator with the FSS model

The open-loop regulator is based on the FSS model already described by equation (1). It must take into account the fact that the FSS considers only active workloads and if a given workload is not CPU-bound, then remaining CPU portions might be consumed by other workloads.

Let's assume that: (i) number N_w of workloads ≥ 2, (ii) number of active workload is changing at time t according to activity state vector $A^t = [A^t_1, \ldots, A^t_{Nw}]$ where $A^t_i = 0$ if W_i is not active and $A^t_i = 1$ if W_i is active, i = 1,...,N_w, (iii) each workload is CPU bound and has allocated shares S_i. Following mathematical transformations of equation (1) we obtain equation (4) for shares S_w to be set for workload W_w:

$$\mathbf{S^t}_w = (\mathbf{U_w} \sum^{Nw}_{i\neq s} S_i * A^t_i)/ (1-\mathbf{U_w}) \qquad (4)$$

Where, $\sum^{Nw}_{i\neq s} S_i * A^t_i$ is the disturbance monitored by the manager and is equal to the sum of all active workload shares excluding workload W_w at time interval t.

5. Solaris 10 case study

This section presents a case study of controlling workloads within the Solaris 10 environment. The implementation used the local control loop, running within a *JIMS Management Agent* on the machine on which the workload was running. The goal of this control loop was to adjust the *project.cpu-shares* resource control to a value which would assure that a given percentage of CPU time would always be available for a given workload.

5.1. FSS control applicability rules

When regulating CPU shares for the project we should check if the OS is fully utilized, because only then does FSS schedule the threads according to assigned CPU shares to specific projects; otherwise CPU resources which are not used are assigned to other workloads. This fact impacts the implementation of the controller. It would be very desirable to add a simple rule which checks if the whole OS is CPU-bound.

Figure 4 depicts the case when a single CPU-bound process is started. Unfortunately, as can be noticed, output from *prstat* increases gradually over the period of more than one minute because *prstat* shows incremental CPU usage for the workload and moreover the measured value does not reach 100 %. This behavior impacts the controller interval. The value must be adequate with respect to the time required by FSS to adapt itself to specified shares and variable load. It is necessary to point out than *prstat* is the only utility which reports CPU utilization per project. It is, however, possible to check immediate, total CPU utilization using the *vmstat* Solaris utility, as shown in Fig 4. For that reason implementation of the rule which checks if the system is fully utilized uses the Solaris *vmstat* command, referring to the *kernel statistics* (KSTAT) interface. It may happen that under a fully utilized system some application threads are not properly preempted. Such a scenario occurred in the test depicted in Fig. 5. In this case, the JIMS monitoring thread responsible for acquiring monitoring data wasn’t scheduled properly by the OS and if the data are not effectively acquired, the closed-loop controller calculates the shares as 0. Such a situation might be observed when the OS is saturated i.e. the running queue length is greater than the number of CPUs. As a workaround, a simple rule is used, which checks if the size of the vector which stores data about CPU usage of controlled project is greater then zero. This rule is evaluated at each regulator interval and if the result is negative, that operation is cancelled but only for the current interval.

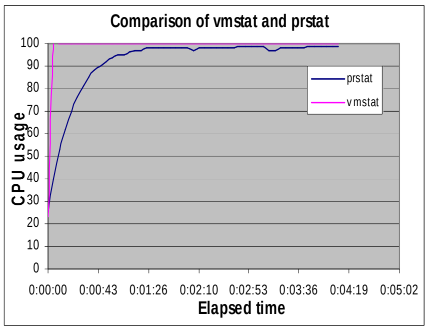

Figure 4. Comparison of *vmstat* and *prstat* monitoring tools.

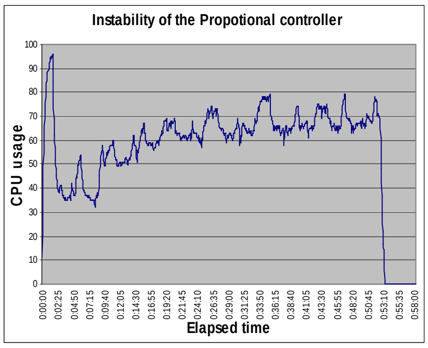

Figure 5. Instability of the closed-loop Proportional (P) controller caused by irregular monitoring thread scheduling.

When considering the case with a *variable* disturbance we should take into account a situation where there is only one CPU-bound workload and the whole CPU is assigned to that workload. In such a case it makes no sense to run the control loop. A rule might be implemented on the basis of the FSS model, which assigns CPU according to share value, considering other active workloads (Listing 1). This implementation returns a list of workloads and the controller may check whether the list contains a specific workload.

```
List getCPUBoundWorkloads () {
    List workloadsList = new ArrayList();

     /** Get active projects for which number of processes
         and CPU usage is greater than zero */
    List activeProjects = getActiveProjects();

     /** Calculate the sum of shares of these projects */
    int sumOfShares = 0;
    foreach (projectId:activeProjects) {
       sumOfShares += getProjectShares(projectId);
    }

    /** Project is to be considered CPU-bound if the current CPU usage is
        greater or equal then its CPU  entitlement according to assigned
        shares */
    float entitlement; float cpuUsage;
    foreach (projectId:activeProjects) {
      entitlement = getProjectShares(projectId)/
                         sumOfShares;
      cpuUsage = getCpuUsage(projected);
      if ( entitlement <= cpuUsage ) {
        workloadsList.add(projected);
    }
  }
  return workloadList;
}
```

Listing 1. Implementation of the rule which finds the list of the current CPU-bound workloads.

Another factor in the case with a variable disturbance is a situation when the CPU share for a controlled workload is not properly calculated. The explanation is very simple – namely when JIMS monitoring data are stored in a vector, some of them are acquired at a time when only controlled workload is active and is assigned nearly the entire CPU. Such data dilute the mean calculated value and if the value is bigger then the goal, shares are decreased instead of increased. The solution to the problem is to use a decaying average, similar to the Jacobson [14] algorithm used in the TCP/IP Protocol to smooth measured values.

5.1 CPU control experiments

This section reports a preliminary study of the closed-loop controller with rules proposed in the previous section. Experiments were performed under Solaris 10 running on a Sun Blade B100 (1GB RAM, CPU SPARC 650 Mhz) board. The goal of control was to *assure a constant allocation of the processor in conditions of variable load* e.g. a given project is guaranteed to use 70% of CPU regardless of the number of other active projects workload and disturbances (also other workloads). The *nspin* application, provided by the Solaris Resource Manager tools, was used. There were two kinds of disturbances: *variable* and *constant* (Fig. 7). A *variable* disturbance was generated with a period of 90 seconds. A *constant* disturbance was activated after the controlled workload reached the steady state (considering the fact it was the only CPU-bound workload, it reached close to 100% CPU usage). CPU consumption due to monitoring and control activity performed by the JIMS Management Agent depicted in Fig. 6 shows that this overhead is not substantial.

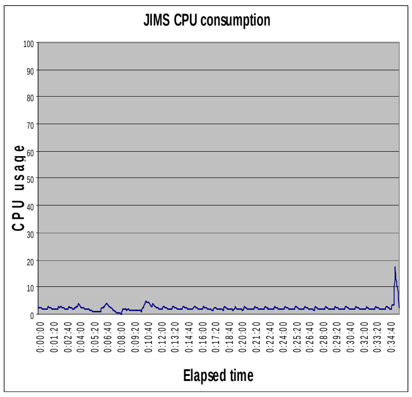

Figure 6. JIMS Management Agent CPU usage.

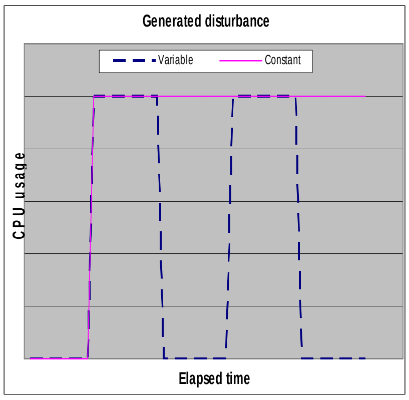

Figure 7. Disturbance generated during tests.

Fig. 8 presents the case where only one CPU-bound workload is started in the selected project, at the beginning. After a few seconds, when CPU usage increases to 95%, two other CPU-bound workloads are started in other projects, which results in a drop of CPU usage of the selected project. Then, after several more seconds, the P controller is turned on. It changed the share allocation to the controlled project, stabilizing CPU usage at 70%. The experiment was repeated for different values of the K_p coefficient. As shown in a Figure 9 the best results was achieved for $K_p=7$.

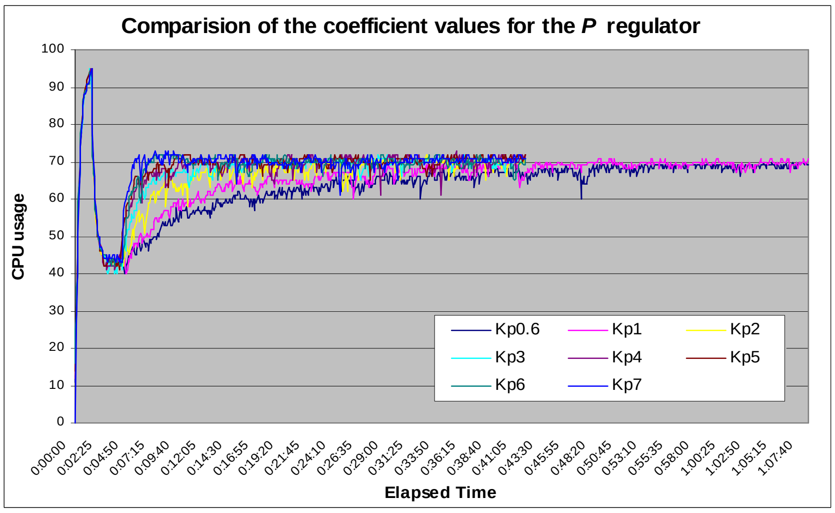

Figure 8. Proportional regulator used for adjusting *project.cpu-shares* resource control for workload with target CPU usage of 70%.

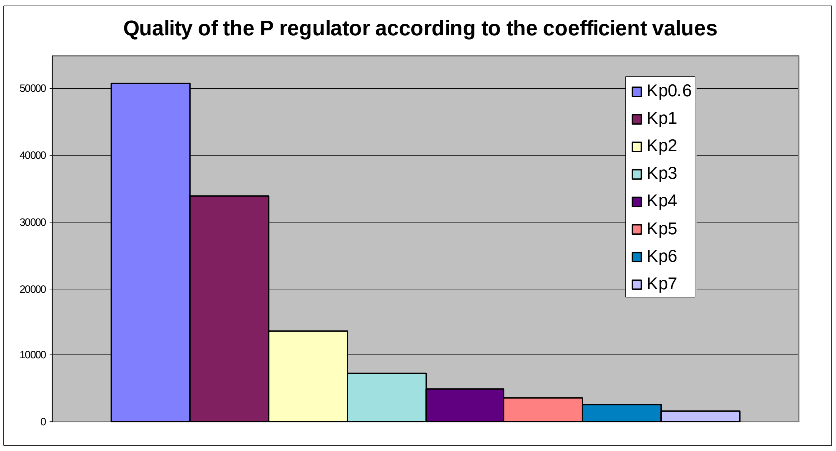

Figure 9. Quality of the Proportional regulator measured using integral of squared error method.

An interesting observation is that despite the complexity of virtualization mechanisms and delay in CPU usage accounting performed by *prstat*, the system can be approximated to the first degree. These results justify the application of P and PI regulators for CPU usage control of the selected project, under variable disturbances shown in Fig. 7, as presented in Fig. 10. It is evident that the smoothing operation performed by Jacobson algorithms significantly improves control quality and stabilizes the system. P and PI regulators provide similar quality of control, as can be seen in Fig. 10. The integral of squared error is equal to *102271* and *100413* for P and PI regulators respectively. Coefficients for the PI regulator were calculated on the basis of the very well know method called step-response analysis [11].

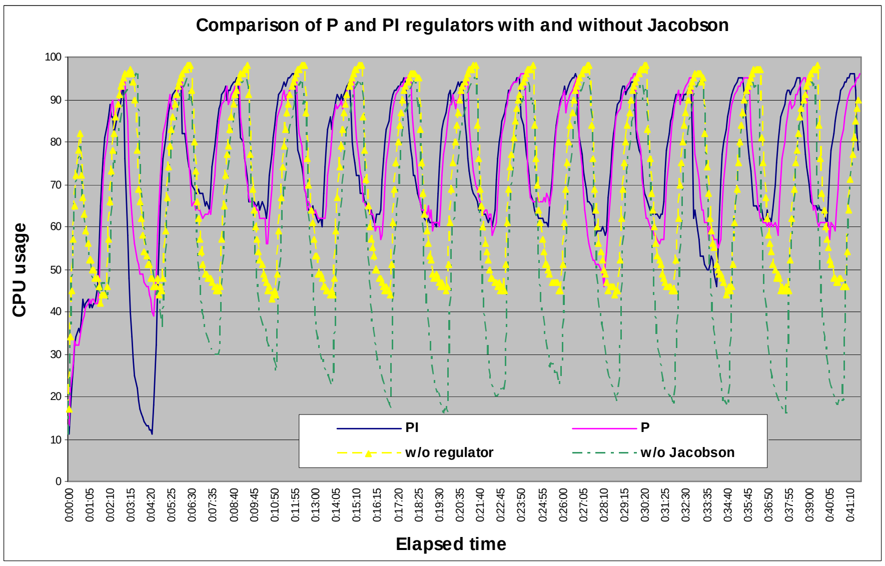

Figure 10. Proportional and Proportional-Integral regulators with the Jacobson algorithm.

6. Conclusions

This paper presents the framework for consolidated adaptive workload management. The primary contribution is organization of the control loop and its implementation with JMX technology, used for exposition mechanisms already supported by modern virtualization technologies. The proposed solution was successfully verified for a simple control policy. It opens a very wide area of research, focused on control strategy selection. The most promising course seems to be the use of a hybrid controller which combines elements of classical control theory with heuristic rules or fuzzy logic. These topics will be the subject of future studies.

Acknowledgments. This research has been partially supported by the Polish Ministry of Education and Science grant no.1583/T11/2005/29.

References

1. Horn, P. Autonomic Computing: IBM's Perspective on the State of Information Technology, October 15, 2001, http://researchweb.watson.ibm.com/autonomic.

2. Kirchstein E., Policy Management for Autonomic Computing: Using three coordinated tools to get managed resources PMAC-ready, April 2005.
3. Zieliński, K., Jarząb, M., Wieczorek, D., Bałos, K. JIMS Extensions for Resource Monitoring and Management of Solaris 10, Advancing Science through Computation - ICCS 2006, LNCS 3994, Springer-Verlag, Berlin/Heidelber, 2006, pp. 1039–1046.
4. Bałos, K., Zieliński, K. JIMS - the Uniform Approach to Grid Infrastructure and Application Monitoring, CGW '04, –Workshop Proceedings, 2005, pp. 160–167.
5. Price D., Tucker A., "Solaris Zones: Operating System Support for Consolidating Commercial Workloads", Proceedings of the 18th Usenix LISA Conference, November 14-19, Usenix, Atlanta, GA, 2004, pp. 241–254.
6. Lageman M., Solaris Containers – What They Are and How to Use Them, Sun Microsystems, http://www.sun.com/blueprints/0505/819-2679.pdf, 2005.
7. Sun Microsystems, Java™ Management Extensions Instrumentation and Agent Specification v. 1.2, JSR 003, available at: http://jcp.org/en/jsr/detail?id=3 (jmx_1.2_spec.pdf), Santa Clara, CA, 2002.
8. "VMware" http://www.vmware.com/.
9. "OpenVZ" http://openvz.org/.
10. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, A. Wareld, Xen and the Art of Virtualization, SOSP 2003.
11. Hellerstein J. L., Diao Y., Parekh S.,Tilbury D. M., Feedback Control of Computing Systems, Wiley-IEEE Press, August 24, 2004, ISBN-13: 978-0471266372.
12. Kay J., Lauder P., A Fair Share Scheduler, Communication of the ACM ,Volume 31, Issue 1, January 1988, ISSN:0001-0782, pp. 44 – 55.
13. Gunther N.J., Solaris System Resource Manager: All I Ever Wanted Was My Unfair Advantage (And Why You Can't Get It!), Dec. 5-10 1999,Computer Measurement Group Conference, Reno, NV.
14. Jacobson, V., "Congestion Avoidance and Control", SIGCOMM, 1988, Stanford, California.