iPOEM: A GPS Tool for Integrated Management in Virtualized Data Centers

Hui Zhang
Robust and Secure System Group
NEC Labs America
Princeton, New Jersey, U.S.A
huizhang@nec-labs.com

Kenji Yoshihira
Robust and Secure System Group
NEC Labs America
Princeton, New Jersey, U.S.A
kenji@nec-labs.com

Ya-Yunn Su
Dept of Computer Science & Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C
yysu@csie.ntu.edu.tw

Guofei Jiang
Robust and Secure System Group
NEC Labs America
Princeton, New Jersey, U.S.A
gfj@nec-labs.com

Ming Chen
Dept of Electrical Engineering & Computer Science
The University of Tennessee
Knoxville, Tennessee, U.S.A
mchen11@utk.edu

Xiaorui Wang
Dept of Electrical Engineering & Computer Science
The University of Tennessee
Knoxville, Tennessee, U.S.A
xwang@utk.edu

ABSTRACT

A fundamental problem that confronts data center administrators in integrated management is to understand potential management options and evaluate corresponding space of the managed system's potential status. In this paper, we present iPOEM, a middleware with GPS-like UIs to support integrated power and performance management in virtualized data centers. iPOEM offers novel system positioning services to enable a declarative management methodology: administrators specify a target location in terms of system performance and power cost, and iPOEM returns the management configurations and operations that are required to drive the system to the target status.

In the core of iPOEM lies an automated management configuration engine exposing two simple APIs: *get_position()* and *put_position()*. We study the relationships between system status and the management configurations in our problem domain, and design a logarithmic configuration searching algorithm for the engine. Several system positioning services are developed atop the engine, including an auto-piloting scheme leveraging sensitivity based optimization technology, and provide intuitive UIs to operation users. The iPOEM prototype is developed atop Usher, an open-source virtual machine management software. The evaluation driven by real data center workload traces shows that iPOEM renders both intuitive usage and efficient performance in the integrated management of virtualized data centers.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICAC-11, June 14-18, 2011, Karlsruhe, Germany.

Copyright 2011 ACM 978-1-59593-998-2/09/06 ...$10.00.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; I.6.3 [**Simulation and Modeling**]: Applications; K.6.4 [**System Management**]: Centralization/decentralization

General Terms

Human Factors, Management, Performance

Keywords

data centers, virtualization, integrated management, simulation based optimization.

1. INTRODUCTION

Integrating power and performance management in virtualized data centers could lead to effective use of IT infrastructures, but requires continuous maintenance of delicate tradeoff to realize the goal under dynamic workload. From workload management point of view, performance management seeks for a balanced load distribution among available servers to tolerate workload dynamics and avoid hotspots, while power management for energy saving brings skewness into server load distribution through server consolidation. The fundamental problem that confronts data center administrators is to understand mutual impact of performance and power management decisions, and have a clear view of potential status space of the managed system under different configuration combinations of the two management components.

In this paper, we present iPOEM (**i**ntegrated **PO**wer and p**E**rformance **M**anagement), a middleware to guide administrators in this tradeoff decision process. iPOEM offers Global Positioning System (GPS)-like user interfaces for its users to configure and control their system: users specify a target location in terms of system performance and power cost, and iPOEM returns the management configurations and operations that are required to drive the system to the destination status. The management configurations include two server load thresholds: CPU_{high}, which the performance

management component enforces the CPU load of all active servers to be no higher than through hotspot elimination; CPU_{low}, which the power management component enforces the CPU load of all active servers to be no lower than through server consolidation. The management operations are the related VM migrations and machine on/off operations that the performance and power management components will execute to maintain server load between CPU_{low} and CPU_{high}.

At the core of iPOEM is a management configuration engine which exposes two API functions for system positioning: *get_position()*, which reports the system status upon input information including VM workload and management configurations, and *put_position()*, which returns the management configurations/operations needed for the system to reach the input specified status. The engine includes a discrete event simulator which models the system and the algorithms used by the performance and power management components, and a logarithmic searching algorithm for the function *put_position()*. The searching algorithm is designed based on the relationship analysis between system status and the management configurations in our problem setting.

We developed the iPOEM prototype based on an open-source virtual machine management framework called Usher [10]. We extended Usher with the implementation of simplified performance and power management components in VMware Dynamic Power Management [18] product, and applied iPOEM for the integrated management of the two components. The iPOEM prototype offers several system positioning services developed atop the configuration engine, including an auto-piloting scheme leveraging sensitivity based optimization technique, and what-if query tools for system position reporting and destination searching under different workload predictions. The evaluation driven by real data center workload traces showed that iPOEM rendered both intuitive usage and efficient performance in the integrated management of virtualized data centers.

The rest of the paper is organized as follows. Section 2 gives the problem formulation and Section 3 presents the iPOEM architecture. The details of the management configuration engine are presented in Section 4, the analysis on the relationships between system status and the management configurations is given in Section 5, and the iPOEM prototype is presented in Section 6. We present the evaluation results in Section 7, the related work in 8, and conclude in Section 9 with future work.

2. PROBLEM FORMULATION

In this section, we first describe two representative algorithms separately for performance and power management in virtualized data centers, then define the integrated management target through three system status metrics, and last present the integrated management as a system mapping problem.

2.1 Performance and Power Management Algorithms

Figure 1 is a simplified performance management algorithm for hotspot elimination in virtualized data centers. When a server is overloaded (e.g., CPU utilization is higher than an upper bound CPU_{max} for several continuous time points), the algorithm will be invoked for hotspot elimination by migrating out enough VMs out of the overloaded server and enforcing the threshold CPU_{high} ($\leq CPU_{max}$) for all servers during the process.

Data Structures:

- VM list with the load distributions and existing VM hosting information $(VM_i, Server_j)$.
- Physical server list including load information, server utilization threshold CPU_{high}.

Algorithm (invoked on an overloading event):

1. sort the VMs in the overloaded server j in decreasing order by the resource demand (calculated as $\mu + 2\sigma$, where μ is the mean and σ^2 is the variance of a VM's history load).
2. place each VM i in order to the best server k in the list which has the new load no higher than the threshold CPU_{high} after hosting i, and yields the minimal remaining capacity.
3. if j's load meets the load threshold CPU_{high} after moving out VM i, terminates the searching process for this server. otherwise, continues the searching for the remaining VMs on j.

Figure 1: A simplified performance management algorithm for hotspot elimination

Figure 2 is a simplified power management algorithm for energy efficiency in virtualized data centers through server consolidation. Periodically (e.g., every a few hours), the algorithm will be invoked for server consolidation by migrating all VMs out of under-utilized server and enforcing the thresholds CPU_{high} and CPU_{low} for the load on all servers.

Performance and power management through controlling server load within a target range $[CPU_{low}, CPU_{high}]$ is a common mechanism, and adopted in many industry products including VMware's Distributed Power Management (DPM) [18] and NEC's SigmaSystemCenter [11], and other research work [19, 2, 17]. Clearly, different settings of the thresholds CPU_{high} and CPU_{low} will lead to different behaviors of the performance and power management components. For example, a low CPU_{high} will lead the performance management component to use more server resource for conservative load balancing, while a high CPU_{low} will lead the power management component to be aggressive during server consolidation process.

2.2 Integrated Management Target Definition

For integrated power and performance management, we define the high-level management target through three system-wide status metrics:

- *Performance cost.* This metric measures the penalty cost due to performance violations when the system runs under a specific management configuration. In this paper, we choose a probabilistic SLA metric [2] which is defined as the percentage of the time in average when servers have CPU utilization higher than the desired target CPU_{high}. It reflects the penalty cost of SLA violations due to server overloading.

Data Structures:

- VM list with the load distributions and existing VM hosting information $(VM_i, Server_j)$.
- Physical server list including load information, server utilization thresholds CPU_{high} and CPU_{low}.

Algorithm (invoked periodically)

1. for the under-utilized servers whose load are lower than the threshold CPU_{low}, sort them in increasing order by the server load.
2. for each under-utilized server j in the order
   (a) place each VM i in j to the best server k in the list which has the new load no higher than the threshold CPU_{high} after hosting i, and yields the minimal remaining capacity. If no such server is available, terminates the searching process for this server and go to next under-utilized server.
   (b) When all the VMs in server j can find a target non-overloaded server to move out, label this server as an empty server to be turned off, and migrate out all the VMs.
   (c) repeat the above steps until no empty server can be found.

Figure 2: A simplified power management algorithm for server consolidation

- *Power cost.* This metric measures the power consumption if the system runs under a specific management configuration. In this paper, it is referred to as the total power (in kWatts) consumed by servers.
- *Operation cost.* This metric measures the frequency of management operations when the system runs under a specific management configuration. In this paper, it is defined as the number of VM migrations that the performance and power management components need to execute for the server load configuration enforcement.VM migrations are an important factor under administrators' consideration when managing virtualized data centers. Many performance/power management tools offer interfaces to control VM migration frequency (e.g., the *migration threshold* in VMware Distributed Resource Scheduler). Frequent VM migrations are not desired since they will clog data center networks and impact negatively on the performance of the applications run in the migrated VMs. Fuirthermore, because of interference due to shared I/O, such migrations also impact the performance of other applications whose VMs run on the same physical hosts. As shown in [6], ignoring these migration costs can have significant impacts on the ability to satisfy response-time-based SLAs.

2.3 Integrated Management: A System Mapping Problem

We formulate the integrated management as a system mapping problem:

- *given a specific configuration on* (CPU_{low}, CPU_{high})*, what's the expected system status (*i.e., *performance cost, power cost, and operation cost)?*

and its dual problem:

- *given an expected system status, what's the specific configuration (*i.e., *the VMs-servers mapping) on (*CPU_{low}*,* CPU_{high}*) leading to it?*

3. IPOEM DESIGN

3.1 Architecture

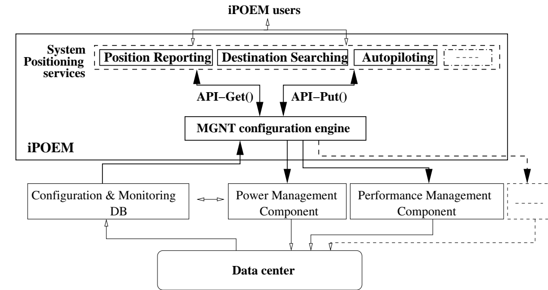


Figure 3: iPOEM architecture.

Figure 3 shows the iPOEM middleware architecture. It takes a layered design and consists of two stacks: management configuration engine, and system positioning services.

3.2 Management Configuration Engine

Like a GPS receiver processor, the management configuration engine consumes run-time system information including VM resource utilization information, server status information, current management configuration settings, and exposes two primitive functions.

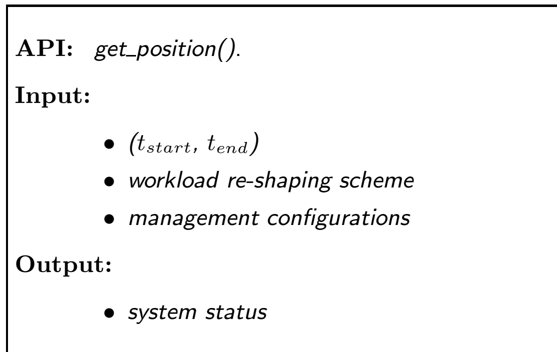


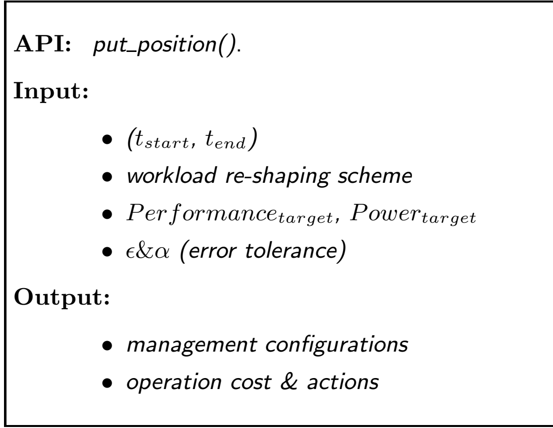
Figure 4: iPOEM primitive function - GET

The first primitive function, called *get_position*, provides the report on where the system would be located in terms of the three cost metrics under certain workload and system settings. As shown in Figure 4, the input and output parameters of the GET API include:

- (t_{start}, t_{end}): this parameter set specifies the interested time period when the VM workload will be used

for system status calculation. Note that the time period can only be the history time when the system/workload monitoring data are available.

- *workload re-shaping scheme*: this parameter provides the option to apply different prediction schemes on the original workload data during the time period (t_{start}, t_{end}). More details on workload re-shaping schemes are described in Section 4.2.
- *management configurations*: this parameter set specifies the management policy settings. In this paper, the configurations include the CPU load control range [CPU_{low}, CPU_{high}]. The default are the actual management configurations during the time period (t_{start}, t_{end}).
- *system status*: this parameter set describes the system location in a virtual coordinate space, like the (longitude, latitude, elevation) geographical coordinate space.


API: *put_position()*.

Input:

- *(t_{start}, t_{end})*
- *workload re-shaping scheme*
- $Performance_{target}$, $Power_{target}$
- $\epsilon \& \alpha$ *(error tolerance)*

Output:

- *management configurations*
- *operation cost & actions*

Figure 5: iPOEM primitive function - PUT

As shown in Figure 5, the second primitive function, called *put_position*, provides the report on how the system should be configured to reach a specific status. Most of the input and output parameters of the PUT API have the same physical meanings as those defined in the GET API. For integrated power and performance management, we limit the high-level system status specification to performance and power cost, and $\epsilon \& \alpha$ specify the allowable errors on the destination location during the searching process.

More details on the management configuration engine will be presented in Section 4.

3.3 System Positioning Services

To end users (e.g., data center administrators), iPOEM presents system positioning services leveraging the two primitive functions. Next we describe three example services mimicking the GPS interfaces in car driving, where the map is on a three-dimensional space and (X, Y, Z) corresponds to the system cost of (Performance, Power, Operation).

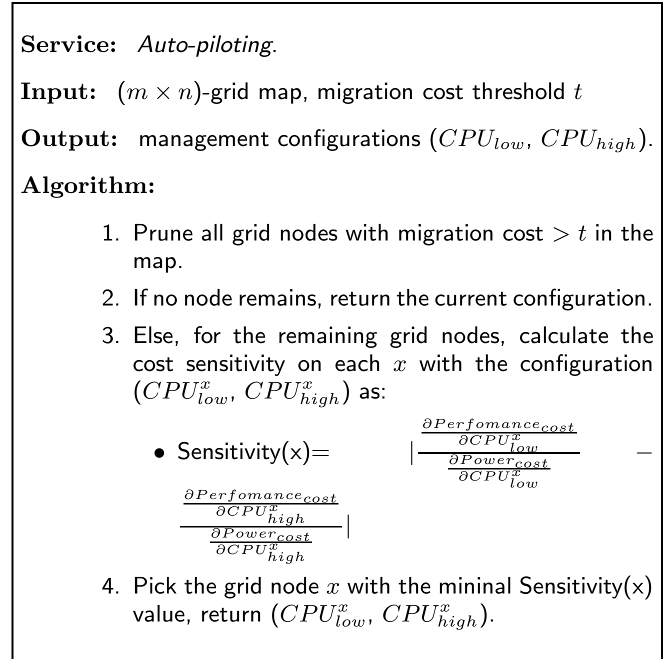
The first service, *position reporting*, gives a visual report on how the system has gone through in the map during the history (e.g., in the past one hour), and several future directions where it could go. It is similar to the tracking function in GPS devices,and built on the *get_position* primitive.

The second service, *destination searching*, provides what-if query interface to automatically generate the management configurations upon a user-specified status point. It also reports a feasibility zone which is defined by the maximally and minimally feasible performance cost and power cost. The status points outside the feasibility zone are not reachable, either due to performance constraint (e.g., the server overloading time $< 5\%$), or due to physical resource limitation (e.g., maximally 200 servers available in the resource pool). It is similar to the navigation function in GPS devices, and built on the *put_position* primitive.

The third service, *auto-piloting* automatically generate an optimal management configuration at the end of each consolidation epoch. The optimality is defined in the context of the following performance-power optimization problem

$$min \text{ Power(configs) s.t. Performance(configs) } \leq P_{th}.$$

where P_{th} is the upper bound of the performance cost.


Service: *Auto-piloting*.

Input: $(m \times n)$-grid map, migration cost threshold t

Output: management configurations (CPU_{low}, CPU_{high}).

Algorithm:

1. Prune all grid nodes with migration cost $> t$ in the map.
2. If no node remains, return the current configuration.
3. Else, for the remaining grid nodes, calculate the cost sensitivity on each x with the configuration (CPU^x_{low}, CPU^x_{high}) as:
   - Sensitivity(x)= $\left| \frac{\frac{\partial Perfomance_{cost}}{\partial CPU^x_{low}}}{\frac{\partial Power_{cost}}{\partial CPU^x_{low}}} - \frac{\frac{\partial Perfomance_{cost}}{\partial CPU^x_{high}}}{\frac{\partial Power_{cost}}{\partial CPU^x_{high}}} \right|$
4. Pick the grid node x with the mininal Sensitivity(x) value, return (CPU^x_{low}, CPU^x_{high}).

Figure 6: iPOEM Auto-piloting scheme

As shown in Figure 6, the auto-piloting scheme chooses the optimal (CPU^x_{low}, CPU^x_{high}) configuration under a migration cost constraint, and is based on a map of $(m \times n)$-sized grid. Each grid point corresponds to the system status position for some (CPU^x_{low}, CPU^x_{high}) configuration and a specified workload. Specifically, we partition the feasibility zone provided by the destination searching service equally into $(m-1)$ sub-ranges along the power cost dimension, and for each of the m power cost points, use the *put_position* primitive to get n points which have different performance and operation cost. The minimal sensitivity difference criteria on the two dimensions is a necessary condition for the optimal solution, and the proof can be referred to [9].

3.4 Discussions

While not an internal iPOEM component, monitoring information calibration is a critical part of the management

framework to ensure the correctness of iPOEM outputs. Its main function is calibrating the monitoring information scrambled by noise, errors, and co-hosting interference. We are pursing a signal processing based solution to this problem, and do not further discuss it in this paper.

4. MANAGEMENT CONFIGURATION ENGINE

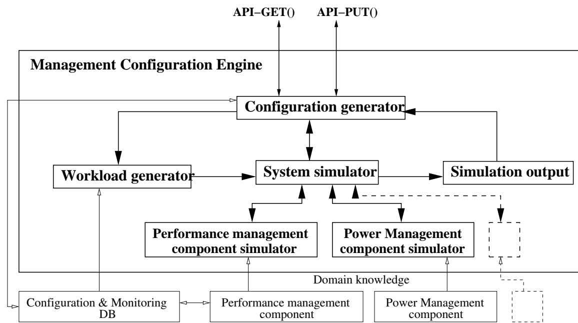


Figure 7: Management configuration engine architecture.

Figure 7 shows the iPOEM management configuration engine architecture. Next we describe the components in details.

4.1 Configuration Generator

This component drives the configuration engine. It takes calls to the two primitive functions, and decides how to transfer them into internal subtasks in the engine and schedule the execution. It contains an efficient searching algorithm for the solution of PUT API calls.

Figure 8 describes the searching algorithm inside the function *put_position()*. Its main steps include the procedure to find CPU_{low} that leads to the desired $Power_{target}$ and the procedure to find CPU_{high} that leads to the desired $Performance_{target}$. Note a ($Performance_{target}$,$Power_{target}$) input might not be a feasible status; we first filter status inputs with the feasibility zone information, and return the feasible status point closest to the input target if it is within the zone. Those details are skipped in Figure 8 for simplicity.

Note that there may exist multiple configurations to reach a desired system status. The searching algorithm is deterministic and will terminate for the first such configuration encountered; we do not seek for optimizations such as minimizing the operational cost in this paper. The searching overhead will be evaluated through the algorithm complexity analysis in Section 5, and through the query response time measurement in Section 7.3.2.

4.2 Workload Generator

This component takes a specified time period as input, and provides embedded schemes to re-shape the VM load information of that period in the monitoring database. The iPOEM prototype inlcudes two example schemes. One scheme takes input a load change factor in percentage (e.g., +20% means increasing each VM's load in record by 20%), and another scheme runs a regression-based load prediction algorithm [14] to output a predicted load on each VM. The output of the workload generator upon each re-shaping scheme is new VM load data in time series to represent speculated new workload scenarios.

4.3 System Simulator

This component is a discrete-event simulator to output the estimated system status and detailed management operations. It answers what-if questions on the system status under different workload scenarios. It simulates system status on server-level, and contains data structures and logic functions to simulate per-server resource utilization information based on the VM workload input from the workload generator. It also includes event logs which lead to the invocation of management component simulators. For example, a detection of server overload during the system simulation process could invoke the performance management simulator, which reacts with the corresponding management operations that the system simulator will execute; a timer register could trigger periodic invocations of the power management simulator to execute server consolidations. The simulated server performance information will be analyzed to output the system status report, so will be on the simulated management operations.

4.4 Management Component Simulators

These components encode the algorithm/scheme details of the performance and power management components. They interact with the system simulator to output the expected management operations when the workload is replayed. In the prototype, we implemented the performance and power management algorithms described in Section 2.

5. ANALYSIS

In this section, we analyze the relationships between the two system status metrics defined in Section 2.2 and the management configurations (CPU_{low}, CPU_{high}).

In the following, we assume a homogeneous system, and the workload remains the same for different configuration settings.

THEOREM 1. *$Performance_{cost}(CPU_{high})$ is a non-decreasing function of CPU_{high}.*

PROOF. Let's keep all the other system settings (including CPU_{low}, the workload) the same, and take two choices on CPU_{high}, CPU^s_{high} and CPU^l_{high}, where $CPU^s_{high} < CPU^l_{high}$. Let $Performance_{cost}(CPU^s_{high})$ be the performance cost under the management methodology described in Section 2, and map^s be the consequent vm-server map produced with the configuration CPU^s_{high}. Clearly, map^s will be a valid load placement for the new configuration CPU^l_{high} from performance management point of view. Let map^l be the consequent vm-server map produced under the new configuration CPU^l_{high}, the difference from map^s to map^l will be caused by the power management component, which may push some server's load higher through server consolidation in map^l. As the elimination of one server (freed after consolidation) with load no more than CPU^l_{high} will create at least one server with the load higher than CPU^s_{high}, $Performance_{cost}(CPU^l_{high})$ could be no less than $Performance_{cost}(CPU^s_{high})$. □

This theorem says that in a homogeneous system, given the same workload, the performance cost typically increases

API: *put_position()*.

Input: t_{start}, t_{end}, workload re-shaping scheme, $Performance_{target}$, $Power_{target}$, $\epsilon\&\alpha$ (error tolerance).

Output: management configurations (CPU_{low}, CPU_{high}), operation cost & actions.

Algorithm:

1. Assign $CPU_{low} = CPU_{max}$, $CPU_{high} = CPU_{max}$;
2. $(Performance_{cost}, Power_{cost})$ = *get_position*(CPU_{low},CPU_{high}, t_{start}, t_{end}, workload re-shaping scheme);
3. If $Power_{cost} > Power_{target}$, then a subset of servers has been turned off forcedly to meet $Power_{target}$;
   (a) (VM-server map, resource inventory) = *forced_down*($Power_{target}$, t_{start});
   (b) $Power_{cost} = Power_{target}$, $CPU_{low} = CPU_{max}$;
4. Else; //start binary searching for CPU_{low}
   (a) $CPU_{left} = CPU_{min}$, $CPU_{temp} = CPU_{low}$, $CPU_{right} = CPU_{low}$;
   (b) while($CPU_{left} < CPU_{right}$)
   (c) $CPU_{temp} = \frac{CPU_{left}+CPU_{right}}{2}$;
   (d) $(Performance_{cost}, Power_{cost})$ = *get_position*(CPU_{temp},CPU_{high}, t_{start}, t_{end}, workload re-shaping scheme);
   (e) If ($Power_{cost} > Power_{target} + \frac{\epsilon}{2}$)
   (f) $CPU_{left} = CPU_{temp} + 1$; $CPU_{low} = CPU_{temp}$;
   (g) Else if ($Power_{cost} < Power_{target} - \frac{\epsilon}{2}$)
   (h) $CPU_{right} = CPU_{temp} - 1$; $CPU_{low} = CPU_{temp}$;
   (i) Else; //find the configuration for CPU_{low}
   (j) $CPU_{low} = CPU_{temp}$; break;
5. $CPU_{left} = CPU_{low}$, $CPU_{temp} = CPU_{high}$, $CPU_{right} = CPU_{high}$;
   (a) while($CPU_{left} < CPU_{right}$)
   (b) repeat binary searching for CPU_{high}
6. return (CPU_{low}, CPU_{high}), and the corresponding operation cost & actions.

Figure 8: iPOEM destination searching algorithm

along with higher CPU_{high}. Intuitively, the lower is CPU_{high}, the more balanced is the load, which leads to lower performance violations.

THEOREM 2. *$Power_{cost}(CPU_{low})$ is a non-increasing function of CPU_{low}.*

This theorem says that in a homogeneous system, given the same workload, the power cost usually decreases along with higher CPU_{low}. Intuitively, the higher the CPU_{low}, the more aggressive is load consolidation in the system, and therefore leads to less power consumption. The proof is similar to that for Theorem 1 and we skip it in this paper.

The monotonicity relationships between system status and the management configurations lead to the correctness of the configuration searching algorithm used in *put_position()*.

COROLLARY 1. *iPOEM destination searching algorithm finds the status destination in $O(\log R)$ steps, where $R = CPU_{max}$-CPU_{min}, is the load control range.*

The proof is straightforward for the two-stage binary searching algorithm given the system properties described above, and skipped in this paper. Note the algorithm starts with the maximal CPU_{low} and CPU_{high}, and searches in the order of the target CPU^*_{low} followed by CPU^*_{high} due to the constraint $CPU_{low} \leq CPU_{high}$. We will present in Section 7.3.5 experiment results to further illustrate those functions.

6. IPOEM IMPLEMENTATION

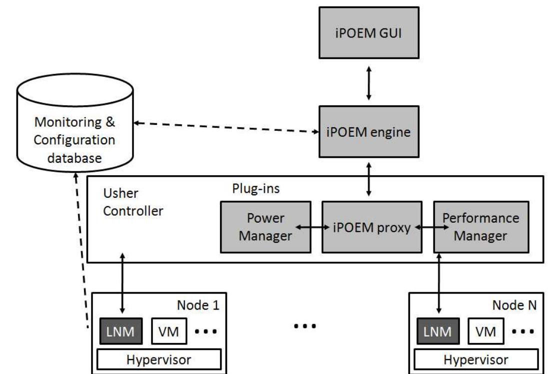


Figure 9: iPOEM prototype implementation.

iPOEM is implementated on Usher, a virtual machine management framework developed by McNett et al [10].

Figure 9 shows the overview of iPOEM. The light-grayed boxes are our addition to the existing Usher framework and we use dark-grayed boxes to color existing Usher components to which we made minor modifications.

6.1 Infrastructure: Usher background

Usher contains three software modules: a centralized controller, a local node management (LNM) for each managed machine, and a client interface. The Usher client library provides API to send virtual machine management requests, such as create or terminate a virtual machine, to the controller that a client application can call. One local node manager runs on each managed physical machine and executes virtual machine requests sent from the controller. A LNM also collects resource utilization data of all VMs running in the LNMand stores them into a centralized monitoring database. We perform per-VM CPU utilization monitoring at the node level with the Xenmon tool [5], and perform per-server power utilization monitoring at the node level with SNMP-enabled intelligent PDUs [1]. In the testbed, the monitoring frequency is set as 30 seconds. The Xenmon CPU overhead is low (1%-2%). and the CPU overhead caused by SNMP clients is almost negligible.

6.2 Power and Performance Management

We implemented the power and performance management algorithms in Section 2 as plug-ins in Usher. The power and performance management components are written in Python under Linux platform, and are totally 1500 lines of code. The coordination between those two components are realized through the iPOEM engine described below.

6.3 iPOEM Engine

The iPOEM engine, an implementation of the architecture in Figure 3, includes a proxy plug-in in Usher and the engine running as a separate process, and they communicate via socket. The iPOEM plug-in registers to the Usher controller for a periodic event and it would be invoked periodically determined by a configurable parameter (we use the same frequency as the performance management component does). On each invocation, the plug-in sends a position reporting request to the iPOEM engine and waits until the engine notifies when the reply is ready. The GUI will periodically pull the engine output from the database for end-user visualization on system position reporting. We implemented the iPOEM engine as a separate process from the Usher framework for ease of code maintenance and re-usability. The engine can be used as an run-time management engine with the auto-piloting service, or can be used as an offline decision supporting tool that users can supply resource utilization data from a different source, such as history data or a public data center trace. The iPOEM engine is written in Python under Linux platform, and are totally 3200 lines of code.

Please note that the iPOEM engine not only can run periodically in real deployments, but also can be triggered by administrators at any time when they want to make a destination search for the optimized system status with specific performance and power constraints. Upon the reply, the administrators can choose whether or not to enforce the engine outputs in the real system.

6.4 GUIs

We implemented a GUI for iPOEM aiming for data center operational usage. The GUI includes two interfaces: one for displaying data center status in the past, now, and the near future; the other provides the destination searching interface, and serves a power-and-performance knob for operators to tune the system to a target system status. The GUI software framework is written in Visual C# with Windows Presentation Foundation (WPF) and executed on Microsoft .NET Framework 3.0. The software consists of 10 classes of program codes, which are 1600 lines in total excluding 3D libraries to implement 3D UI features such as 3D zooming and rotation, and 6 XAML style configurations to define its visual interfaces.

7. EVALUATION

7.1 Data Center Workload Traces

Our evaluation is based on a large set of data center workload traces. The trace file includes the resource utilization data of 5,416 servers from the IT systems of ten large companies covering manufacturing, telecommunications, financial, and retail sectors. The trace records the average CPU utilization of each server in every 15 minutes from 00:00 on July 14th (Monday) to 23:45 on July 20th (Sunday) in 2008. Among them, 2,525 of the servers have the hardware information including the processor speed (in MHZ) and processor/core number. We use the traces on those 2,525 servers in the evaluation. Workload characterization on the traces is skipped due to space limit, and interested readers refer to [3] for details.

7.2 Methodology

We run the iPOEM prototype as an offline engine. It is driven by the data traces stored in the monitoring database, and emulates the integrated management in a virtualized data center hosting the 2, 525 servers as VMs. In the evaulation, we replay their CPU load traces as the corresponding VM load monitoing data.

The offline engine reads the traces and configuration parameters periodically, and the management parameters are

- The performance manager is invoked every 24 data points (i.e., 6 physical hours) while the power manager is invoked every 48 data points (i.e., 12 physical hours). The slow management frequencies are caused by the 15-minute monitoring frequency in the data traces; enough data points are needed to drive the iPOEM engine for meaningful output.
- We assume that the SLA threshold is 90%, which means that if the CPU utilization of a server is larger than 90% we regard it as a performance violation.
- The default $<CPU_{low}, CPU_{high}>$ setting is <40%, 80%>.
- The physical servers in the simulations are homogeneous with the CPU spec as: 3GHZ Quadra-core (the most common CPU model in the traces).
- We assume power consumption per server is either 0 (power-off mode) or 200Watts (power-on mode), simplified on the power model profiled in the local testbed.

7.3 Results

7.3.1 iPOEM GUI - Position Reporting

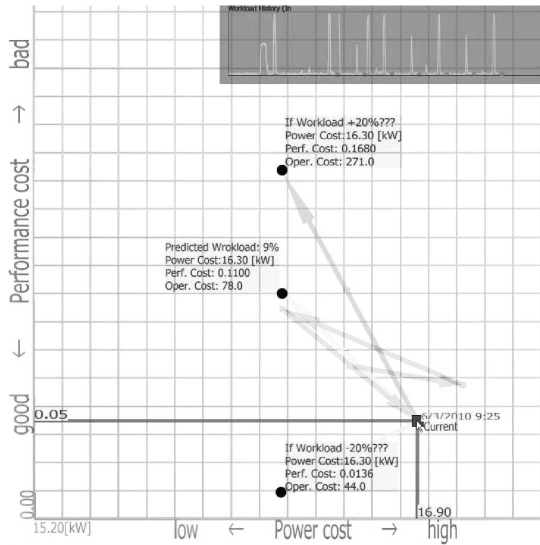


Figure 10: iPOEM GUI - Position Reporting.

Figure 10 shows a snapshot of the first iPOEM GUI interface for position reporting. In this figure, the position of a dot in the two coordinates represents the power cost and performance cost respectively. Figure 10 shows the current system status in a square dot, and three round dots showing three possible system status in the near future upon three workload scenarios: when the current VM workload would increase by 20%, when the current VM workload would decrease by 20%, and when the workload would be like what the regression analysis predicted. Note all the future system status points have the same power cost, which is the power cost after the server consolidation when the power management component would be invoked at the time this snapshot is generated.

7.3.2 iPOEM GUI - Destination Searching

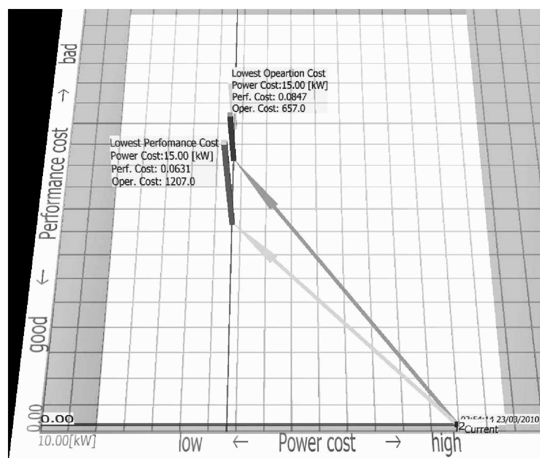


Figure 11: iPOEM GUI - Destination Searching.

Figure 11 shows a snapshot of the second iPOEM GUI interface for destination searching. The prototype uses the past 12-hour workload for the inputs of the function *put_position*, and takes user input for a new target power cost which the adminstrator would like the system to consume but do not know exactly how to configure to reach it. The GUI agent will then send multiple searching requests to the configuration engine, each specifying a target status of the input power cost and an unqiue performance cost within the feasibility zone. The engine searches the configuration space to find possible settings to satisfy the requests. In this particular example of Figure 11, the operator liked to know where the system would be positioned if the operator challenged the system to reduce the power cost to 15.0 kWatts from 21.20 kWatts in the current status. The configuration engine recevied 15 searching requests each with the target status (x%, 15.0 kWatts), where $x = 1, 2, \ldots, 15$. Two possible choices are high-lightened in Figure 11 among the returned configurations, one with the minimal performance cost and the other with the minimal operational cost. Both are represented by two bars; the coordinate position of each bar in the map represents the power cost and performance cost respectively, and the height of the bar represents the operational cost. The details of management operations such as VM migrations are also returned from the management configuration engine, but not shown here due to space limit.

In the case when a target system status is not feasible to reach, the GUI shows the closest state the system can reach and the corresponding configuration parameters. As an example in Figure 11, the gray colored zones in the map represents the infeasible areas where the system cannot reach.

7.3.3 iPOEM engine performance

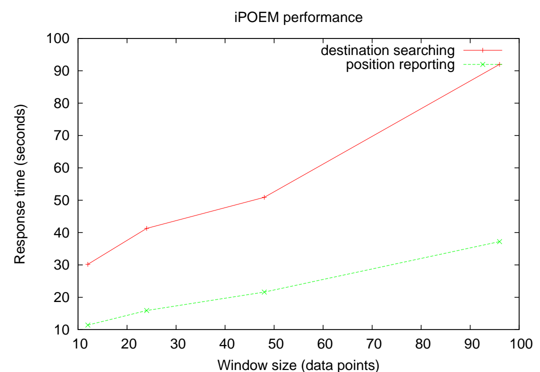


Figure 12: iPOEM engine performance: response time.

Figure 12 shows the performance of iPOEM engine in terms of the response time to requests for the position reporting and destination searching services. The position reporting service sends requests periodically where each request asks for refreshing the current system status and its three potential future destinations; the workload is based on the history data in a certain time window, where the window size is measured by the number of monitoring data points. As shown in Figure 12, the iPOEM engine's response time to the position reporting service increases slowly with the time window size. For the time window size of 48 data points, it takes in average around 22 seconds for the iPOEM engine

to generate all data needed in the position reporting service. Figure 12 also shows the average response time of the destination searching service when a user randomly picks locations in the map to query. Similarly, it increases slowly with the time window size, and typically takes tens of seconds on a system with 2525 VMs.

7.3.4 Auto-piloting output

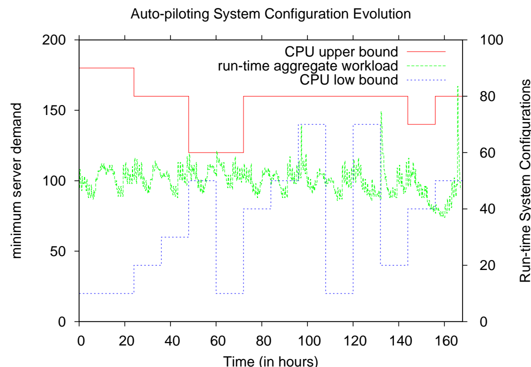


Figure 13: Auto-piloting management configuration evolution.

We also ran the auto-piloting service on the traces, and compared its output with three static (CPU_{low}, CPU_{high}) configuration schemes: $(10\%, 10\%)$ for best performance cost (in time percentage of SLA violations), $(10\%, 90\%)$ for best migration cost (in number of VM migrations), and $(90\%, 90\%)$ for best power costi (in kWatts). Table 1 shows the average cost (and the standard deviation) results of the three schemes, and clearly auto-piloting has good trade-off among the three costs. Figure 13 shows the evolution of the management configurations [CPU_{low}, CPU_{high}] (the curve "CPU low bound" for CPU_{low}, and the curve "CPU upper bound" for CPU_{high}) output by the auto-piloting service along with the time; in addition, the run-time aggregate workload (sum of all VMs' CPU load, normalized into server number) is plotted here for the background information. As we can see, the auto-piloting service adaptively moved to power-efficient mode upon stable workload by narrowing the range [CPU_{low}, CPU_{high}]; once in a while a load spike appeared, which triggered the auto-piloting service moving back to performance-oriented mode immediately, and the range [CPU_{low}, CPU_{high}] became wide again.

7.3.5 Relationships between System Status and Configurations

Next, we show the system status as a function of (CPU_{low}, CPU_{high}) during a certain time period of the trace (the day of July 15th). We set $CPU_{max} = 90\%$, $CPU_{min} = 10\%$, and enumrate all (CPU_{low}, CPU_{high}) combinations with the step size of 10% and the condition $CPU_{low} \leq CPU_{high}$. This leads to 45 different configuration candidate sets.

Figure 14 shows how the performance cost changes as we use different CPU_{low}, CPU_{high}. As described in Section 3.2, this metric measures the performance penalty cost if the system were configured by the input parameters. Since CPU lower bound should be no more than the upper bound, there are no values on the left half of the surface. We can see that, generally speaking, the smaller the CPU bounds, the less the

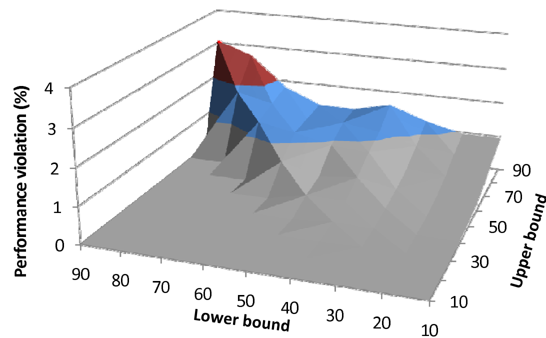


Figure 14: The system status as a function of (CPU_{low}, CPU_{high}): performance cost

performance violations are. The reason is intuitive since the more conservative we place load over servers, the less likely that the server has performance violations.

We skip the similar results on power cost due to space constraint.

8. RELATED WORK

Integrated power and performance management is a key requirement in data centers today. On device level, Li et al. [8] proposed performance-directed energy management for main memory and disks, and designed scheduling algorithms that adjust the values of certain thresholds used in device control policies to meet performance goals; Riska et al. [12] proposed a novel model and an algorithm that manages to successfully explore feasible regions of power and performance on individual disks. On cluster level, Steinder et al. [15] built a state-of-the-art performance manager to achieve significant power savings without unacceptable loss of performance; Sankar et al. [13] explored the use of multiple dynamic knobs and used sensitivity-based optimization to guide the setting of these knobs at runtime to maximize energy efficiency in storage system. On data center level, Kumar et al. [7] proposed vManage, a solution to loosely couple platform and virtualization management and facilitate monitoring and management coordination in data centers; Chen et al. [4] extended their previous work by designing a novel thermal-aware load metric and integrated it into the management of data centers on application performance, power and cooling. Compared to the previous work, our work is focused on the declarative management methodology and its enabling technologies including the system positioning primitives.

The model of the management configuration engine can be mapped into a simulation-based optimization problem, which is the finding of an optimal configuration for a stochastic function with an unknown structure. Often heuristics are applied to improve some configuration instead of really performing optimization in practice. Optimization of discrete event simulation models includes region exploration and exploitation algorithms to find promising local minima in the search space [20]and ranking and selection algorithms [16]. Compared to the previous work, our work leverages the domain knowledge on the system properties under investigation, and designs a fast searching algorithm instead ad hoc heuristics.

Scheme	*Performance cost*	*Power cost*	*Migration cost*
Auto-piloting	1.55% (4.38%)	20.88 (9.9)	37.7 (53)
Static - (10%, 10%)	0	96 (13.5)	342 (510)
Static - (10%, 90%)	2.26% (3.66%)	24.68 (5.08)	23.4 (55.4)
Static - (90%, 90%)	4.46% (3.93%)	16.1 (2)	118.4 (407.2)

Table 1: Comparison of Auto-piloting and three static configuration schemes.

9. CONCLUSIONS & FUTURE WORK

This paper presents iPOEM, an integrated power and performance management middleware in an virtualized infrastructure. iPOEM features novel system positioning services for simple and intuitive user interactions, and includes a management configuration engine to enable data center declarative management.

In data center management, application performance, thermal, cooling, and per-core power management are just a few to name in addition to the performance and power management addressed in this paper. Taking those into the iPOEM framework leads to an explosive growth of the system state space, and how to make iPOEM scalable to those extensions is a challenge work.

In Cloud computing, it is interesting to extend iPOEM to support mashup services for customized tenant management on cloud resources. Tenants may introduce their own performance and power cost functions atop the resource monitoring data, and redefine operation cost with specialized performance and power management functions. This can lead to the creation of virtual maps customized for individual tenants, and enable flexible cloud resource management.

Acknowledgements

We thank the anonymous reviewers and our shepherd, Vana Kalogeraki, for their valuable feedback on this paper.

10. REFERENCES

[1] Raritan px-1000 metered ipdu. http://www.raritan.com/products/power-management/px-1000/.

[2] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing SLA violations. In *IM '07*, pages 119–128, Munich, Germany, 2007.

[3] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira. Coordinated energy management in virtualized data centers. Technical Report UT-PACS-2010-01, University of Tennessee, Knoxville, http://pacs.ece.utk.edu/EffectiveSize-tr.pdf, 2010.

[4] Y. Chen, D. Gmach, C. Hyser, Z. Wang, C. Bash, C. Hoover, and S. Singhal. Integrated management of application performance, power and cooling in data centers. In *NOMS '10*, 2010.

[5] D. Gupta, R. Gardner, and L. Cherkasova. Xenmon: Qos monitoring and performance profiling tool. Technical Report HPL-2005-187, HP Labs, 2005.

[6] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. A cost-sensitive adaptation engine for server consolidation of multitier applications. In *Middleware '09*, pages 9:1–9:20, New York, NY, USA, 2009.

[7] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan. vManage: loosely coupled platform and virtualization management in data centers. In *ICAC '09*, pages 127–136, New York, NY, 2009.

[8] X. Li, Z. Li, Y. Zhou, and S. Adve. Performance directed energy management for main memory and disks. *Trans. Storage*, 1(3):346–380, 2005.

[9] D. Markovic, V. Stojanovic, B. Nikolic, M. A. Horowitz, and R. W. Brodersen. Methods for true energy-performance optimization. *IEEE Journal of Solid State Circuits*, 39(8):1282–1293, August 2004.

[10] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: An extensible framework for managing clusters of virtual machines. In *LISA '07*, pages 167–181, November 2007.

[11] NEC SigmaSystemCenter. Integrated virtualization platform management software. http://www.nec.com/global/prod/sigmasystemcenter/.

[12] A. Riska, N. Mi, E. Smirni, and G. Casale. Feasibility regions: exploiting tradeoffs between power and performance in disk drives. *SIGMETRICS Perform. Eval. Rev.*, 37(3):43–48, 2009.

[13] S. Sankar, S. Gurumurthi, and M. R. Stan. Sensitivity based power management of enterprise storage systems. In *MASCOTS*, pages 93–102, 2008.

[14] R. H. Shumway. *Applied statistical time series analysis.* Prentice Hall, Englewood Cliffs, 1988.

[15] M. Steinder, I. Whalley, J. E. Hanson, and J. O. Kephart. Coordinated management of power usage and runtime performance. In *NOMS*, pages 387–394. IEEE, 2008.

[16] J. R. Swisher, S. H. Jacobson, and E. Yücesan. Discrete-event simulation optimization using ranking, selection, and multiple comparison procedures: A survey. *ACM Trans. Model. Comput. Simul.*, 13(2):134–154, 2003.

[17] A. Verma, P. Ahuja, and A. Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware'08*, New York, NY, USA, 2008.

[18] VMware. *VMware Distributed Power Management Concepts and Use.* http://www.vmware.com/files/pdf/DPM.pdf.

[19] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *NSDI '07*, Cambridge, MA, April 2007.

[20] J. Xu, B. L. Nelson, and J. L. Hong. Industrial strength compass: A comprehensive algorithm and software for optimization via simulation. *ACM Trans. Model. Comput. Simul.*, 20(1):1–29, 2010.