BrownMap: Enforcing Power Budget in Shared Data Centers

Akshat Verma, Pradipta De, Vijay Mann, Tapan Nayak, Amit Purohit, Gargi Dasgupta, Ravi Kothari

BrownMap: Enforcing Power Budget in Shared Data Centers

Akshat Verma, Pradipta De, Vijay Mann, Tapan Nayak, Amit Purohit, Gargi Dasgupta, and Ravi Kothari

IBM Research-India

Abstract. In this work, we investigate mechanisms to ensure that a shared data center can operate within a power budget, while optimizing a global objective function(e.g., maximize the overall revenue earned by the provider). We present the *BrownMap* methodology that is able to ensure that data centers can deal both with outages that reduce the available power or with surges in workload. *BrownMap* uses automatic VM resizing and Live Migration technologies to ensure that overall revenue of the provider is maximized, while meeting the budget. We implement *BrownMap* on an IBM Power6 cluster and study its effectiveness using a trace-driven evaluation of a real workload. Both theoretical and experimental evidence are presented that establish the efficacy of *BrownMap* to optimize a global objective, while meeting a power budget for shared data centers.

1 Introduction

The inevitability of power outages in a power grid due to the complex nature of the grid is slowly being realized [2]. Overloads on the electrical system or natural calamities like storms etc. can disrupt the distribution grid, triggering a failure. In the year 2007, more than 100 significant outages had been reported in North America [19]. The gap between the creation of new power generation units and the increasing growth-driven demand in emerging economies make the problem even more acute in developing countries. A 2008 report by Stratfor [15] indicates that the growth in GDP outpaces growth in power production in China by a factor of 5 and in India by a factor of 2. As a result of the power shortages, enterprises depend on back-up generators to deal with outages. However, the backup power available is typically much smaller than the demand leading to another electrical condition called power *brown-outs*, i.e., reduction in the power available to a data center.

A brownout is a temporary interruption of power service in which the electric power is reduced, rather than being cut, as is the case with a blackout. Brownouts may happen due to the reduced power available in the back-up generators as well as grid failures. A brownout may exhibit itself as lower available power (in watts) as well as voltage reduction, for example, the voltage may drop from 120V to 98V or less. Among the 100 power outages in North America in 2007, 33 of them led to brownouts [19]. These brownouts can last anywhere between minutes to few hours depending on their severity and enterprises have to deal with them. In

this work, we investigate the problem of handling brownouts for virtualized data centers that host multiple customers (e.g., clouds) with minimal loss in revenue.

1.1 Triggers for Brownout

The increased focus on energy awareness necessitates demand response planning for handling brownouts in data centers. Brownouts may be triggered in a data center due to : (i) blackouts handled by back-up generators with reduced power supply (ii) unplanned grid failures due to natural calamities leading to reduced power(iii) deliberate brownouts undertaken by the grid authority when it is sensed that grid frequency is falling below safe limits (iv) Planned Demand Response by the data center. Planned Demand Response is emerging as a monetary tool to operate data centers in a cost effective fashion. Power grids have started differential pricing to deal with low frequency problems. For example, the KSE grid in India charges Rs 570 per KiloWattHour (KwH) for any additional unit consumed over the allocated quota if the supply frequency dips below 49 Hz, whereas the regular charges at 50 Hz are Rs $4/KwH$ [24]. In developed countries, multiple demand response initiatives are available to large data centers that may lead to significant cost benefits for the data center [8]. The Pacific Gas and Electric (PG&E) company offers multiple demand response alternatives with financial incentives to large customers during periods of peak load [18]. The *PeakChoice* program allows a customer to decide the power reduction the company is comfortable contributing and advance notice it needs. The Critical Peak Pricing (CPP) program gives an option of lower energy rates on non-peak days (up to a factor of 3) in exchange for higher rates on peak load days with a prior day notice. The Demand Bidding Program (DBP) provides an incentive to reduce a customer's electric load according to a bid. For each peak load event, the datacenter may elect to submit or not submit a bid via PG&E's Internet-based energy management system. Hence, demand response or brownout planning is emerging as an important tool for low cost datacenters.

1.2 Brownout and Clouds: Challenges

Virtualization and cloud computing have emerged as the enabling technologies for driving significant cost reductions in large data centers. One of the main drivers of the rapid adoption of shared data centers or clouds is the utility computing model where customers can scale up or down their resource usage on demand. To support this notion of utility computing, while keeping the costs down, the cloud infrastructure needs to ensure that resources could be moved from one customer to another in response to the customers' changes in demand. Hence, dynamic resource allocation to different customers on a shared infrastructure is the key to operating a cloud at low cost. Dynamic resource allocation has the additional challenge of providing customers the abstraction of an infinite resource pool. Further, the presence of differentiated applications (e.g., customers with different SLAs) on the shared resource pool, adds additional complexity to the resource allocation task.

Brownout management adds another layer of complexity to the resource allocation problem. To manage a brownout, power needs to be viewed as a separate resource and allocated to individual applications in the data center in a way

such that the overall power budget is met, while maximizing the revenue of the provider. Existing resource management techniques (e.g., CPU allocation) can not be directly applied for allocating powers to applications as (i) different servers in a data center may have different power efficiency (ratio of compute capacity per unit Watt) and hence power allocation of an application depends on the type of the server hosting it, (ii) power consumed by one server may differ widely depending on the placed applications (iii) servers have a large static power that needs to be accounted to hosted applications, and hence the power accounted to an application depends on other applications hosted on the same physical server. Hence, existing CPU/memory allocation techniques can not be used to allocate power to applications in clouds.

Creating a power budget for an ensemble of blade [20] or rack servers [12] has been addressed earlier. The goal in these cases is to find the aggregate peak power consumption and use it as a budget. If the actual power exceeds the estimated budget, a throttling mechanism is used at each server. The presence of virtualization and differentiated applications make application-unaware server throttling of limited use in shared data centers and clouds. Multiple virtual machines running the cloud applications with different SLAs (and utility) may be co-located on a common server and a server-based power management mechanism is unable to differentiate between these applications. Further, due to the limited dynamic power range of a server [26, 25], the power reduction due to throttling may even fail to meet the power budget.

1.3 Contribution

In this work, we present *BrownMap*, a fine-grained resource allocation mechanism, which can dynamically expand or shrink resource allocations of individual applications in a shared data center. *BrownMap* is designed to deal with brownouts, where a substantially lower power budget may be available due to a power outage. It can also help a cloud provider to deal with power surges that arise due to workload variability. The contribution of our work is two-fold:
(i) We present the design and implementation of the runtime *BrownMap* power manager. The power manager uses a distributed monitoring and reconfiguration framework. Our design has minimal monitoring overheads and reconfigures the server cluster to meet the power budget in a very short duration. The *BrownMap* architecture is robust enough to deal with noise in monitored data and scales well with the size of the server cluster.
(ii) We present the *BrownMap* placement methodology to find the configuration that maximizes the utility earned by the applications, while meeting the power budget. The methodology uses a novel divide and conquer methodology to break the hard power budgeting problem into *Server Selection*, VM Resizing and *VM Placement* sub-problems. We use an iterative procedure that leverages the nature of the power and utility curves to find a configuration that is close to the optimal for most practical scenarios. On a real testbed using production traces, we show that *BrownMap* meets the reduced power budget in the order of minutes and can bring the power down by close to 50% for a 10% drop in utility.

2 Model and Preliminaries

We now formally define the brownout problem addressed in this paper. Consider a virtualized data center (cloud or traditional) with upto M servers S_j hosting N applications A_k. Each application may either be single component or multi-tiered. Each application component is run in a dedicated virtual machine (VM) or logical partition (LPAR). We use the terms VM and LPAR interchangeably in this work. The data center experiences a brownout for the next T hours with the available power reduced to P_B. The goal of the *Power Manager* is to re-allocate resources to each application component, migrate virtual machines between servers, and switch servers to low power active states (e.g., using DVFS) or lower power idle states (e.g, nap mode) or inactive (i.e., switched off) states in order to meet the power budget. Further, we need to ensure that the utility is maximized while meeting the budget. Formally, we need to find an allocation (or VM Size) x_i for each application component i and a mapping y_i^j on each server S_j s.t.

$$\arg\max_{\mathbf{x},\mathbf{y}} \sum Utility(x_i) \tag{1}$$

$$\sum_{j=1}^{M} Power_j(\mathbf{x},\mathbf{y}) < P_B; \quad \sum_i y_i^j {x_i}^i \leq C_j, \quad \forall\ S_j; \quad \sum_{j=1}^{M} y_i^j = 1, \quad \forall\ VM_i \tag{2}$$

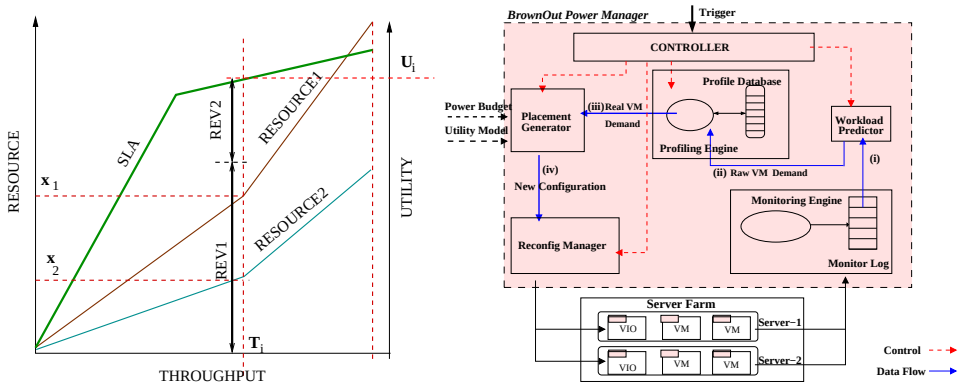
where $Utility(x_i)$ is the utility earned by VM i if it is assigned x_i resources, $Power_j(\mathbf{x},\mathbf{y})$ is the power drawn by server S_j for the given resource assignment and placement, and C_j is the capacity of the server S_j.

2.1 Deriving VM Utility

We have used a utility maximization framework to capture the importance of each application to the shared data center. The utility would usually be computed based on the revenue earned by the data center for the application. A utility based framework is general enough to capture various other objectives like strict priority and fairness. Application SLAs for clouds that provide platform as a service (PaaS) can be expressed as a 'revenue versus resource' curve. SLAs for clouds that provide application or software as a service (AaaS) are captured as a 'revenue versus application SLA metric' curve, where the application SLA metric could be throughput. In this work, we use 'revenue versus throughput' as the SLA since it captures both PaaS and AaaS. Our framework requires a utility function (utility versus resource curve) to be associated with individual virtual machines, which needs to be derived from application SLAs.

For single component applications, the utility of a VM could be derived in a straightforward manner. For each application, the provider can use a few experimental runs to derive a 'resource versus throughput' curve and combine it with the SLA (revenue versus throughput curve) to obtain a 'revenue versus resource' curve for the application. This curve may be used as the utility for the VM as it accurately captures the revenue obtained by the provider for each unit of resource allocated to the application. The provider may then use these utility curves to determine the applications that provide the least loss in revenue if resources are taken away from them.

Traditional data centers as well as clouds also host many multi-tier applications, where each tier is hosted in a separate VM. Consider an example scenario



(a) (b)

Fig. 1. (a) Utility Computation in BrownMap and (b) BrownMap Architecture

with a 2-tiered application in Fig. 1(a). The SLA function captures the revenue derived by the composite application for a given throughput. The $RESOURCE$ function captures the resource consumption of each tier (or VM) for a certain application throughput. Resource functions for each tier can be obtained using monitored data that map resource utilization in each component VM to the application throughput. In cases where an application may have multiple types of requests, the functions are based on average estimates. In order to apply our framework, we need to derive the utility functions for each component from these SLA and $RESOURCE$ functions.

The chosen utility function for each VM should satisfy the following properties:
1. Proportional Allocation: Resource allocation to each component (VM) of an application should be in a manner such that no component becomes a bottleneck. To understand the need for Proportional Allocation, consider the example in Fig. 1(a). If LPAR1 has x_1 resource assigned and LPAR2 has resource assigned greater than x_2, any additional resource assigned to LPAR2 above x_2 does not lead to an increase in the throughput of the application. This is because, unless, the resources assigned to all the VMs hosting the multiple components of a multi-tier application are increased proportionately, the overall application throughput will remain unchanged and any additional resources assigned to only a few VMs (in this case LPAR2) will get wasted.
2. Utility Preservation: The total utility obtained by all the VMs hosting a multi-component application for aggregate resource usage x_i should equal the utility of the composite application for a resource allocation of x_i. This is required to ensure that the utility attached to the components of an application reflect the SLA of the composite application.

In order to achieve these goals, we have designed a proportional utility assignment method. We divide the utility derived by the application amongst its components in proportion to the resource used by each component. Hence, in Fig. 1(a), we set the revenue $REV1$ of $Component1$ at throughput T_i as $\frac{x_1}{x_1+x_2} * U_i$. Futher, the utility function (revenue versus resource) attached to VM of $Component1$ has utility $REV1$ at resource x_1. The above assignment ensures Utility Preservation, i.e., sum of the utility of all components of an application for a given resource usage equals the actual utility of the application

at the resource usage. Further, the utility per unit resource consumed for each component is equal ($= \frac{U_i}{x_1+x_2}$ in the example). We later show (Sec. 4) that our optimization methodology uses the above insight to meet the proportional allocation property as well.

3 BrownMap Architecture

We now describe the overall architecture of the *BrownMap Power Manager*.

BrownMap Power Manager computes a new sizing and placement for the Virtual Machines (VMs) for a fixed duration termed as the consolidation interval (e.g., 2 hours). The key modules in the BrownMap Power Manager, as shown in Fig. 1(b), are (i) *Monitoring Engine*, (ii) *Workload Predictor*, (iii) *Profiling Engine*, (iv) *Placement Generator*, and (v) *Reconfig Manager*. The *Monitoring Engine* periodically collects (a) system parameter values from each logical partition (LPAR) as well as the management partition present on each physical server in the shared cloud infrastructure and (b) Application usage statistics, and stores them in the Monitor Log. The *power management* flow is orchestrated by a *Controller*. The *Controller* executes a new flow on an event trigger, which could be either a change in the power budget or the end of the previous consolidation interval. On receiving an event trigger, it invokes the *Workload Predictor*, *Profiling Engine*, *Placement Generator*, and *Reconfig Manager* in the given order to coordinate the computation of the new configuration and its execution. The main steps in the flow are (i) estimation of the resource demand for each VM in the next consolidation interval by the *Workload Predictor*. (ii) update of the VM resource demands to account for VIO resource usage based on the profiles of each application by the *Profiling Engine*, (iii) re-sizing and placement of VMs based on their utility models and a power budget by the *Placement Generator*, (iv) execution of the new configuration by the *Reconfig Manager*. We now describe each component of our architecture separately.

3.1 Monitoring Engine

The monitoring engine collects resource usage statistics for all partitions, including the management partition, on a physical node. For IBM's Power Hypervisor (pHyp) the management partition is called the Virtual I/O (VIO) Server. The monitor agent on each partition collects utilization data for CPU, active memory, network traffic, and I/O traffic, and feeds it back to the monitoring engine. The data is sampled every 30 seconds, and periodically the aggregated log files are pushed to a *Monitor Log*, implemented as a relational database. The script based monitor agent on each partition consumes less than 0.1% of the resource allocated to a partition.

The resource dedicated to a partition can change dynamically under different settings. For example, the number of CPUs allocated to a partition can vary over time. The task of monitor agent is to accurately track the changing resource consumption of the partition, also called "entitlement of the partition". The actual resource usage, for CPU and memory, is percentage utilization with respect to the entitlement. Note that the CPU entitlement for each LPAR in a Power6 virtual environment can be a fraction of the total CPU pool available on the

physical server. Many of the modern processors are also capable of scaling the voltage and frequency in order to optimize power consumption, which is known as Dynamic Voltage and Frequency Scaling (DVFS). If DVFS is enabled, the reported entitlement takes into account the scaled CPU frequency. The resource usage statistics of the VIO are similarly monitored and logged. It is important to note that the VIO performs work on behalf of individual LPARs, which should be accounted back to the LPAR. The profiling engine, discussed later ensures that the LPAR resource usage captures the work done by VIO on its behalf.

3.2 Workload Predictor

The goal of the *Workload Predictor* is to estimate the raw resource (CPU, memory, network, disk) demand for each VM in the next consolidation interval. It has been observed in data centers that some workloads exhibit nice periodic behavior and some do not follow any particular pattern [27]. Periodic workloads can be predicted in longer horizons with significant reliability using a long term forecast. However, the prediction error increases rapidly for non-periodic workloads as the horizon increases. We use a two-pronged strategy in the design of our *Predictor* to handle both periodic and non-periodic workloads. We make a short-term as well as a long term prediction and use both to estimate the resource usage.

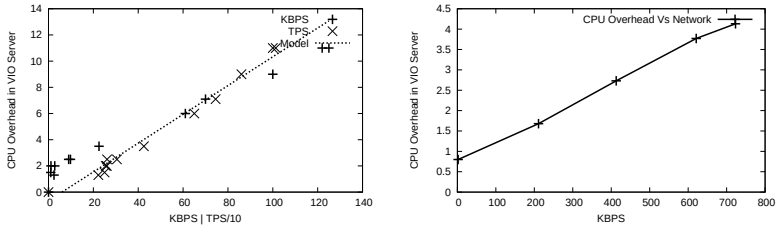
A popular method for deciding periodicity is the auto-correlation function and the peaks in the magnitude spectrum [1]. Once the periodicity for a workload is determined, we use it to make a long term forecast. The short-term prediction is based on polynomial approximation to minimize the least-square error. We then use a weight parameter to give weightage to the long term and short term forecasts. Let us divide the usage history into a sequence of time periods and we consider the last P periods for estimation. Our goal is to forecast the next K usage values based on last n samples of the usage history, where $n = P*p$ (p is the number of samples in each period). The resource demand at $(n+k)$-th interval is predicted as

$$\hat{D}_{n+k} = (1-\alpha) * \frac{1}{P} \sum_{i=1}^{P} y_{i*p+k} + \alpha * f_p(y_n, y_{n-1}, \ldots, y_{n-N_s-1}), \quad k = 1, \ldots, K$$

where y_{i*p+k} are the corresponding usage values at the k^{th} sample of the i-th cycle, f_p is the short term prediction based on last N_s samples and α is the weight parameter. Note that $\frac{1}{P}\sum_{i=1}^{P} y_i$ and $f_p(y_{n-1}, y_{n-2}, \ldots, y_{n-N_s})$ represent the long and short-term components of the forecasted value, respectively. We set α as 1 for workloads without periodicity and 0.5 otherwise. For the resource usage histories we considered, we found that second order is sufficient for reasonable approximation, and the error increases as K increases. Finally, the default value of P is set such that the number of periods cover a week, which has been observed to be sufficient to determine periodicity in data centers [27].

3.3 Profiling Engine

I/O processing for virtual I/O adapters are performed by the Virtualization layer (Virtual I/O Server(VIO) in pHyp and dom0 on Xen) on behalf of individual LPARs. This redirection leads to a CPU overhead for I/O processing [6] that must be accounted back to the LPAR requesting the I/O. As an example, the



(a) VIO Overhead with LPAR Disk Activity (b) VIO Overhead with LPAR network activity

Fig. 2.

CPU overhead in VIO due to an LPAR having a network activity of $600KBps$ on an IBM JS-22 BladeCenter is around 4% of 1 $4.0GHz$ Power6 core (Fig 2(b)). The *Profiling Engine* provides an estimate of such VIO CPU overhead and accounts it to the LPAR in order to create the real resource demand for each LPAR.

The *Profiling Engine* uses a *Profile Database* that captures the relationship between hypervisor CPU overhead and the disk and network activity in an LPAR for each physical server type (Fig 2(a) and Fig 2(b)). This profile is created using calibration runs on each server model in the datacenter. During the *power management* flow, the *Profiling Engine* uses the *Profile Database* along with the network and disk activity demand provided by the *Workload Predictor* to estimate the VIO overhead due to each LPAR. The VIO overhead is then added to the raw CPU demand of the LPAR to estimate the real resource demand for each LPAR in the consolidation interval. The real demand is used by the *Placement Engine* for VM sizing and placement.

The *Profiling Engine* also provides an estimate of power consumption for a configuration that the *Placement Generator* comes up with. The power drawn by a server can not be accurately estimated only from the expected CPU utilization of the server for heterogeneous applications, as it also depends on the nature of the application [25]. The *Profiling Engine* uses WattApp, a power meter that has been designed for heterogeneous applications [16]. WattApp uses power profiles for each application on a server, which is then used to estimate the power drawn by a server for running a mix of applications. The individual power profile for each application is also stored in the *Profile Database*.

3.4 Placement Generator

The *Placement Generator* takes as input the predicted real resource demand (CPU, Memory, I/O) and the application profiles from the *Profiling Engine*, utility models for the workloads, the previous allocation and the user specified power budget. Based on resource demands and the utility accrued from each application, it computes a a new placement map, which specifies which applications reside on which servers and occupy what capacity (i.e. size) of the host server. Further, based on the application and server profiles, the VIO layer is also resized. The power consumed by this new placement map is now within the power budget and maximizes the overall utility. Details of the methodology implemented by the *Placement Generator* are presented in Sec. 4.

3.5 Reconfiguration Manager

The *Reconfiguration Manager* takes as input the new LPAR entitlements and placement provided by the *Placement Engine* and moves the data center to this new configuration in the most efficient manner possible. LPAR migration is an expensive operation (1 to 2 minutes for active LPARs) and it is important to minimize the time taken to reconfigure the data center. Further, since LPARs may both move in or out of a server, it is possible that there may not be available resources for an LPAR moving in till some LPARs are moved out. This may create temporary resource capacity issues leading to failures during reconfiguration. The goal of the *Reconfiguration Manager* is to (a) minimize the total time taken to reconfigure and (b) avoid any temporary capacity failures due to migration. The *Reconfiguration Manager* spawns a new process for each server that has at least one LPAR being migrated from it. This allows the reconfiguration to scale with the number of servers involved in the reconfiguration. In order to overcome any capacity issues during migration, we reduce the reservation of an LPAR being migrated, before migration. This ensures that there is no resource allocation failure when an LPAR is migrating to a target server. The LPAR is resized back to its correct resource allocation, as soon as the target server has finished migrating any LPARs that are moving out of it. If a server has no LPARs running on it, the *Reconfiguration Manager* either switches it off or moves it to *Nap* mode [11], if available. For active servers that are underutilized, an appropriate power-state is selected using DVFS.

4 BrownMap Sizing and Placement Methodology

We now present the *BrownMap* sizing and placement methodology implemented by the *Placement Engine*. There are three sub-problems which are tackled to reach the overall goal of revenue maximization under limited power budget. These are: (i) Selection of active servers that meet the power budget, (ii) Resource Allocation for each VM (x_i) on the available capacity of the servers and (iii) Placement of VMs on active physical servers (y_i^j). The resulting problem is an NP-hard problem, as bin packing becomes a simple case of this problem.

In order to understand the problem, observe that the optimization problem evaluates two functions, namely *Power* and *Utility* as a function of VM sizes (x_i) and their placement (y_i^j). As noted in [25], power consumption by a server is not determined solely by the server's utilization but is also dependent on the nature of applications running on the server. Hence, estimating the power drawn by a heterogeneous mix of applications in a shared cloud is a challenging problem. Since a closed form for an objective function, viz. power, does not exist, off-the-shelf solvers can not be used. The Utility function is again a derived measure and is dependent on the SLA parameter of the application. Typical utility functions would satisfy the law of diminishing marginal returns and may be approximated by a concave curve. The value of the SLA parameter depends on the resource assigned to the application and is typically a convex function (e.g., Fig. 4(b)). Hence, the nature of the utility curve with the resource is a function with potential points of inflection making it a difficult problem to solve.

We now present an outline of our *BrownMap* methodology that solves this problem using an iterative procedure.

4.1 Outline of BrownMap

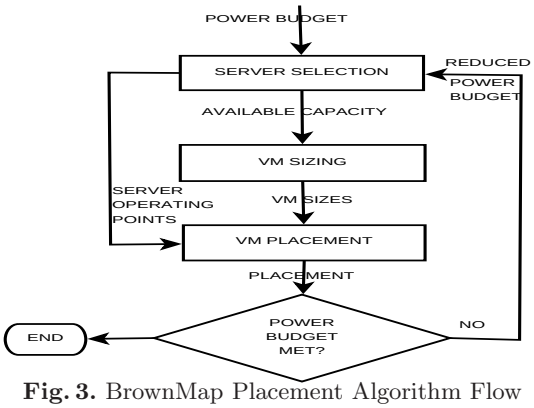


Fig. 3. BrownMap Placement Algorithm Flow

The *BrownMap* methodology is based on a 'divide and conquer' philosophy. We divide this hard problem into three sub-problems, namely Server Selection, VM Sizing and VM Placement. One may note that the feasible space for each sub-problem depends on the solution of other sub-problems. For example, the *Server Selection* problem can predict whether the power budget can be met, only if the placement and sizing of the applications are known. To address this problem, we assume a fixed solution for the other two sub-problems, while finding the best solution for each sub-problem. We then iterate over the solution till we can find a placement and sizing that meets the power budget. Our methodology leverages a recent power modeling work Wattapp [16], which is able to predict the power drawn by a heterogeneous mix of applications running on a shared cloud, for convergence. The overall iterative flow of *BrownMap*, as shown in Fig. 3, consists of the following steps.

- Server Selection: In this step, we use the available power budget to pick the most power efficient servers within the power budget for an average application. We leverage the Order Preservation property from an earlier work [25] that allows us to rank servers with respect to power efficiency in an application oblivious manner. Once we identify the active servers, we compute the aggregate server capacity available to the applications.
- VM Sizing: This step takes the available capacity as input and computes the best VM size (x_i) for each application that maximizes the utility earned within the available capacity.
- VM Placement: In this step, we use the servers and their operating points and VM sizes to compute the best placement (y_i^j) of VMs on servers. We use the history-aware iDFF placement method presented in an earlier work [25]. $iDFF$ is a refinement of the First Fit Decreasing bin packing algorithm that also minimizes the number of migrations. For further details of this step, the reader is referred to [25]
- Iterate: If the Placement method is able to place all the applications, we use WattApp [16] to estimate the power drawn. If the estimated power exceeds

the budget or the applications can not be placed in the previous step, we iterate from the server selection step with a reduced set of servers.

4.2 Server Selection

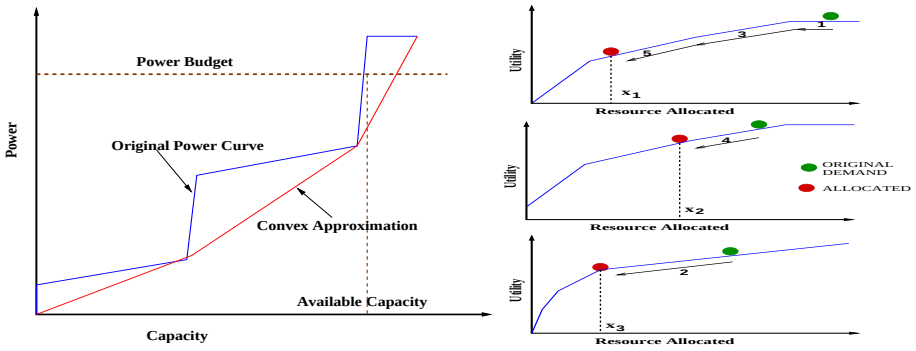


Fig. 4. (a) Server Selection and (b) VM Sizing Method

We define the *Server Selection* problem as finding a subset of servers and their operating points such that the power drawn by the servers is within a power budget. Further, the total server capacity available for hosting VMs is the maximum possible within the power budget. Note that the power drawn by a server is not determined solely by the CPU utilization but also on the application mix running on the server [9], [25], [26], [16]. Hence, the power drawn by a set of servers can not be represented as a closed function of the capacity used, as it depends on factors other than server capacity. However, we have noted in [25] that an ordering can be established between different servers based on their power efficiency for any fixed application and this ordering holds across applications.

We use the *Ordering property* to order servers and create a power vs capacity curve in Fig. 4(a). We replace the original curve with a convex approximation and find the server capacity that meets the power budget. The convex approximation is employed for the iterative convergence. We use this selection of servers and operating points for VM sizing and placement. Once we get an allocation of VMs to servers and their sizes, we estimate the actual power consumed by the servers using the WattApp meter [16]. If the estimated power drawn exceeds the power budget, we move down on the convex power curve and iterate again.

4.3 VM Sizing

The *VM Sizing* problem takes as input an aggregate server capacity available to host a set of VMs. Further, for each VM, a model of utility as a function of the SLA parameter of the VM and a model of SLA parameter versus resource assigned to the VM is taken from the Pre-Processing step. We use the utility and resource models to create a model of utility versus resource allocated for each server. We start *VM Sizing* by allocating to each VM the maximum resource required by it in the next consolidation interval. We then iteratively take away

resources from the VM with the least slope of the Utility-Resource curve (or the VM which has the least drop in utility for a unit decrease in resources allocated). To take the example in Fig. 4(b), we first take away resources from the first VM (with the least slope). In the next step, the third VM has the least slope and we reduce its resource allocation. The *VM sizing* method terminates when (a) the total capacity used by the VMs equals the capacity given by the server selection process and (b) the selected point on each curve is either a corner point or the slope of all the non-corner points are equal (Fig. 4(b)). The first property ensures that we meet the power budget assuming the *Server Selection* process is accurate. The second property ensures that the overall utility drawn by the VMs within the power budget can not be increased by making small changes. It is easy to see (proof omitted due to lack of space) that the resultant solution has the following optimality property.

Theorem 1. *The VM Sizing Method finds a resource allocation that is locally optimal. Further, if the utility versus capacity models for all VMs are concave, then the resource allocation is globally optimal.*

We note that the VM sizing for one VM is independent of other VMs. Hence, if an application has multiple components, with each component hosted in a separate VM, the sizing may lead to resource wastage. A desirable property is that each component of an application should be sized to achieve the same throughput. In order to achieve this, we add another condition to the convergence. If two VMs compete for a resource and have the same slope on the utility-capacity curve, we assign the resource to the VM with a lower achieved SLA value. This property coupled with the utility breakup for multi-tier applications (Sec. 2.1) leads to the following proportional assignment property between VMs belonging to a multi-VM application.

Property 1. Proportional Allocation Property: For a multi-tier application, the VM sizing allocates resources to all the components of an application in a way that they lead to the same application throughput.

4.4 Iterative Procedure

The *Iterative procedure* takes the computed placement and uses the *WattApp* power meter to estimate the power consumed by the placement. If the estimated power meets the power budget, the *Reconfiguration Manager* is triggered to reconfigure the VM placement in the cloud. If the estimated power is more than the budget, we iterate from *Server Selection* with a lower budget.

The *Brownout* problem has two optimization objectives: (i) power minimization and (ii) utility maximization. Minimization problems that have a convex objective function and maximization problems with a concave objective function lead to a fractional optimal solution easily. Hence, we have converted the power curve in the *Server Selection* problem to a convex approximation. In case the utility-capacity curve for all the applications is concave, this implies that the iterative procedure would converge to a solution that minimizes power and maximizes utility for a fixed server capacity. Further, the solution is also within the power budget and is close to the fractional optimal solution. We have formalized the above proof sketch to obtain the following result. The detailed proof is omitted for lack of space.

Theorem 2. *For any instance I of the brownout problem (Eqn. 1), consider the modified problem I', which replaces the power function by its convex approximation. The* BrownMap *sizing and placement methodology leads to an integral approximation of the LP-relaxation of I' for concave utility functions.*

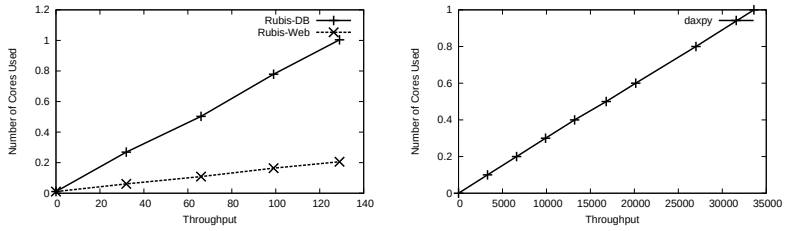
5 Experimental Evaluation

5.1 Prototype Implementation

We have implemented *BrownMap* to manage a cloud consisting of 8 IBM Power6 JS-22 blade servers hosted on an IBM Bladecenter H chassis. Each blade has 4 IBM Power6 4.0 GHz cores with $8GB$ of RAM installed and is connected to a Cisco Network Switch for network access. The blades use 2 Gbps Qlogic Fiber Channel ports to connect with a SAN consisting of an IBM DS 4800 storage controller with 146 SCSI disks. The *BrownMap* power manager is deployed on a dedicated management server. The management server is an IBM Power5 1.5 GHz machine with 1 GB of RAM running Linux kernel 2.6. In order to completely automate the management, the management server has password-less ssh enabled with all the LPARs and VIOs. Monitoring agents are deployed on all the LPARs and VIOs. The management server uses the BladeCenter Advanced Management Module to get power data about the managed blades.

5.2 Experimental Setup

We now describe the experimental setup used to evaluate *BrownMap*.



(a) Rubis (b) daxpy

Fig. 5. Resource Vs SLA Models

Applications and Traces Used We have deployed 2 different applications on our cloud testbed. The first application deployed in our testbed is *Rubis* [21] that simulates an auction site like ebay. *Rubis* is a two-tiered application with a web front end and a backend database server. We denote the web tier as *Rubis-Web* and the database tier as *Rubis-DB*. *Rubis* executes the standard browse and buy mix that is supplied with the application. Our second application is *daxpy*, a BLAS-1 HPC application [7]. *daxpy* takes batch jobs as input and executes them. We run two variants of *daxpy*; namely *daxpyH* as a high priority application and *daxpyL* as a low priority application.

We have instrumented both the applications and created models for throughput versus consumed CPU resource. The CPU resource is expressed in terms of the number of Power6 4.0 GHz cores for normalization across all LPARs (refer Figure 5). We use three different utility models for the applications. *daxpyH* is given a utility function that makes it the highest priority application in the testbed. *daxpyL* has a utility function with the least utility per unit amount of resource

consumed. We attach a utility function for *Rubis* that is intermediate between *daxpyH* and *daxpyL*. The exact utility functions for the three applications for a given throughput (ρ) are (i) *daxpyH*: Util(ρ) = $\frac{5\rho}{33600}$, (ii) *daxpyL*: Util(ρ) = $\frac{\rho}{33600}$, (iii) *Rubis*: Util(ρ) =$\frac{3\rho}{129}$. A natural interpretation of the utility functions is that for the same resource used, *daxpyH*, *daxpyL* and *Rubis* get utility in the ratio of 5 : 1 : 3. Note that 33600 is the throughput for *daxpy* and 129 is the throughput of *Rubis* at 1 core resource usage. We created drivers for each application that takes a trace as input and creates workload for the application in a way that simulates the utilization given by the traces. We used utilization traces collected from the production data center of a large enterprise for our study. More details about the traces are available in an earlier work of ours [27].

Competing Methodologies We compared *BrownMap* against a few other methodologies that an administrator can use to deal with brownouts .
Baseline: The first methodology termed as *Baseline* mimics the scenario where a *Brownout Manager* does not exist. The methodology is useful as it achieves the maximum utility without any resource restriction and can be used to compare with the utility achieved by other methodologies.
Server Throttling: This methodology throttles all the servers in the shared infrastructure to enforce the power budget. However, this approach was unable to bring the power down significantly in order to meet the budget in most cases.
Proportional Switchoff: This methodology switches off enough number of servers to meet the power budget. Further, it reduces the resources allocated to the VMs in a proportional manner. Finally, it migrates LPARs from inactive bladeservers to active blade servers.

We compare all the methodologies with respect to their ability to meet the power budget and the utility achieved, while meeting the power budget. Further, we also study their throughput and migration overheads.

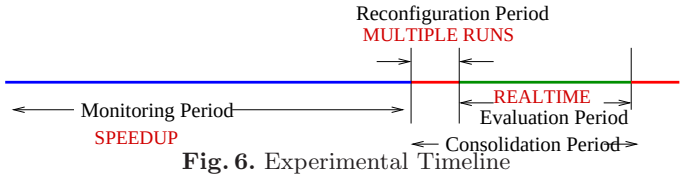


Fig. 6. Experimental Timeline

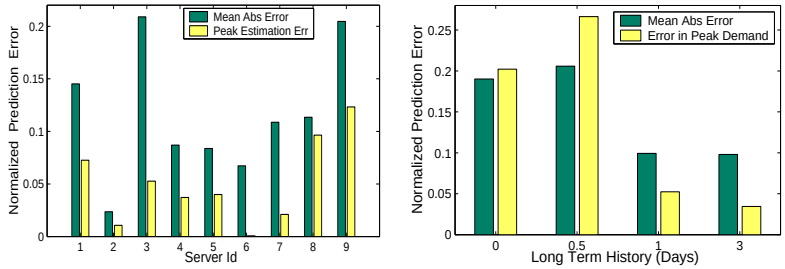
Experimental Timeline Each experimental run in our study has three different phases, as shown in Figure-6. In the *Monitoring Period*, we collect monitoring data for the applications and the servers. Once sufficient historical data is available, we periodically run the *BrownMap* methodology and reconfigure the data center. Each consolidation interval can thus be broken down into *Reconfiguration Period* during which we transition to the new configuration. This is followed by the *Evaluation Period* during which the configuration is allowed to run. Once the evaluation period is over, a new consolidation period starts.

The traces evaluated had weekly periodicity requiring the *Monitoring Period* to be 7 days or more. The reconfiguration activity (VM resizing and migration) depends on a large number of factors and should be repeated multiple times

for the results to be meaningful. As a result, each experimental run takes an inordinately large time. In order to speed up the experiments, we divided each run into different parts. The *Monitoring Period* was speeded up using the already available trace data. The *Reconfiguration Period* was repeated multiple times in real time and each measure is a statistical average. The *Evaluation Period* was run in real-time and the throughput obtained by each application was measured.

5.3 Experimental Results

We performed a large number of experiments to evaluate *BrownMap* under a wide variety of settings. We now report some of our important observations.



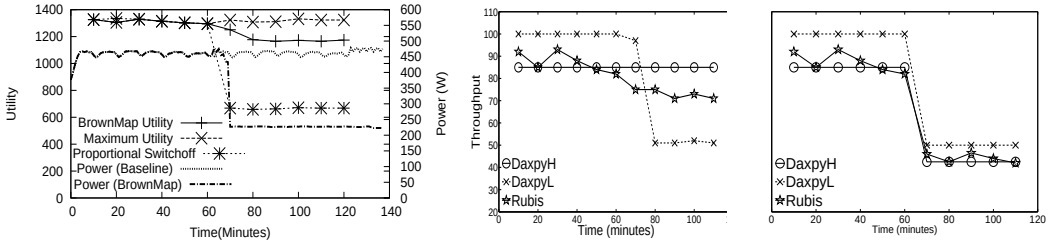
(a) (b)

Fig. 7. (a) Error in Prediction of Demand and Peak Demand (b) Impact of Monitoring period on Prediction Accuracy ($\alpha = 0.3$, $N_s = 2.5Hrs$)

Prediction Accuracy *BrownMap* uses the prediction made by the *Workload Predictor* to determine the expected load in the next consolidation interval. The *Workload Predictor* makes an estimate of the workload intensity for the entire consolidation interval and the maximum workload intensity during the period is taken as the demand of the LPAR. Hence, we study the accuracy of the complete prediction as well as peak demand prediction in Fig. 7(a). An interesting observation is that the peak workload (max) during the evaluation period is a more stable metric and can be predicted to a greater accuracy than predicting the complete workload for the evaluation period. We observe that the error in predicting max is bounded by 10%, which is quite acceptable. We also observe that the prediction accuracy for both the time-varying workload during the evaluation period and the maximum workload improves with increase in monitoring data available, exceeding 95% for a history of 3 days (Fig. 7(b)). An interesting observation we make is that using a very low history for periodic traces may introduce more errors in long term prediction than not using any history at all (0.5 days history has higher error than 0 history).

Comparative Evaluation The first scenario we evaluated was on a 2-blade cluster. We created 6 LPARs on the blades and deployed two instances of $daxpyH$ and $daxpyL$ each. On the remaining two LPARs, we installed the *Rubis*-DB and *Rubis*-Web applications. The cluster during normal operation was consuming 500 watts. We simulate a brownout situation and send a trigger to the *Controller* that the budget had changed to 250 watts.

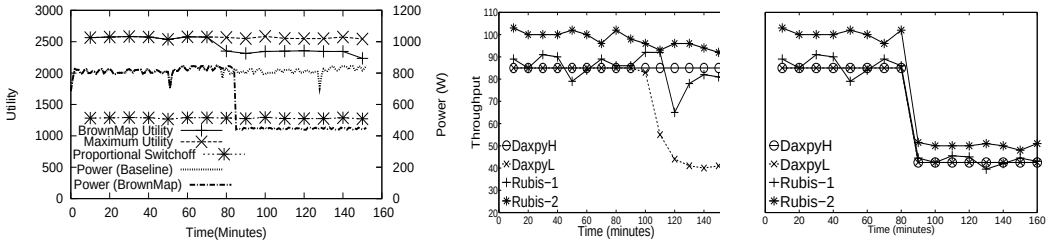
We study the new configuration executed by *Power Manager* in Fig. 8(a). The *Power Manager* resizes the LPARs and moves them to $blade1$, resulting in a drop



(a) (b) (c)

Fig. 8. Consolidating 2 blades with a power budget of 250W: (a) Comparative Power Consumption and Utility Earned. The Power drawn by Proportional SwitchOff is omitted as it closely follows BrownMap. (b) Normalized Throughput achieved by BrownMap (c) Normalized Throughput achieved by Proportional SwitchOff

in power. We observe the power drawn and utility obtained using (i) BrownMap, (ii) Baseline and (iii) Proportional SwitchOff. The utility drawn by *Baseline* indicates the maximum utility that can be earned if no power management actions are taken. We observe that *BrownMap* is able to meet the power budget without any significant drop in utility (about 10% drop from maximum). On the other hand, *Proportional SwitchOff* incurs a 45% drop in utility from the maximum to meet the power budget.



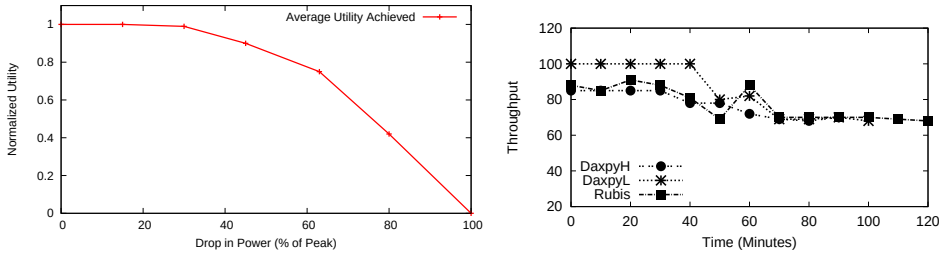
(a) (b) (c)

Fig. 9. Consolidating 4 blades with a power budget of 500W: (a) Comparative Power Consumption and Utility Earned. The Power drawn by Proportional SwitchOff is omitted as it closely follows BrownMap. (b) Normalized Throughput achieved by BrownMap (c) Normalized Throughput achieved by Proportional SwitchOff

To understand how *BrownMap* is able to reduce power significantly without incurring a significant drop in utility, we observe the throughput achieved by each application in Fig. 8(b). We note that *BrownMap* is able to keep the throughput of *daxpyH* at the same level as the one before the brownout happened. On the other hand, it takes a lot of resources from *daxpyL* leading to a significant drop in throughput. The *BrownMap* methodology does not take any resources away from *Rubis*. However, since the overall system utilization is now higher, it leads to a marginal drop in the throughput of *Rubis*. In contrast, as seen from Fig. 8(c), *Proportional SwitchOff* drops the throughput of all the applications by 50%. Hence, *BrownMap* carefully uses the utility function to assign more resources to applications with higher utility per unit resource consumed (Fig. 8(a)). This allows *BrownMap* to meet the power budget with minimal drop in utility.

We next investigate a 4 server cluster to investigate how *BrownMap* deals with larger number of servers. We create 12 LPARs with 4 instances of *daxpyH* and

daxpyL each. On the remaining 4 LPARs, we install 2 instances of Rubis, i.e. 2 LPARs with *Rubis-DB* and 2 LPARs with *Rubis-Web*. We again observe that *BrownMap* adapts to the reduced power budget quickly without a significant drop in utility (Fig. 9(a)). *Proportional SwitchOff* again has to sacrifice a significant amount (close to 50%) of utility to achieve the power budget. The brownout is handled by carefully taking away resources from the low priority application, thus meeting the power budget with no more than 10% drop in utility. We also evaluated *BrownMap* with a 6 node cluster that conformed to the above observations. For lack of space, we do not report those results.



(a) (b)

Fig. 10. (a)Utility earned by BrownMap with change in Available Power. Both Power and Utility are normalized. (b) Fairness Scenario: Normalized Throughput Achieved by all applications

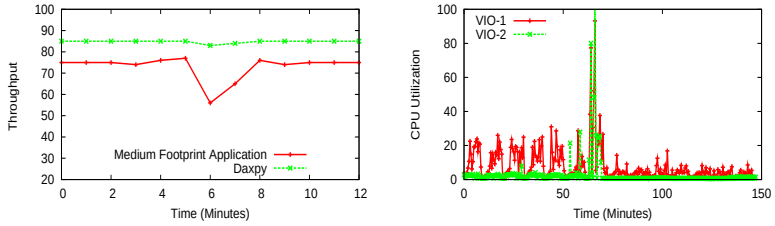
Drop in Utility with Change in Power Budget In real data centers, the reduction in available power due to a brownout varies widely. Brownouts that happen because of increased demand may reduce the available power by 10% or 20% whereas a major outage may reduce available power by as much as 75%. Hence, we next vary the power budget and study the ability of *BrownMap* to deal with brownouts of differing magnitude in Fig. 10(a).

We observe that *BrownMap* is able to meet the budget for upto 50% drop in power with less than 10% drop in utility. This is a direct consequence of the fact the *BrownMap* first takes resources away from the lowest priority applications. These applications do not contribute much to the overall data center utility and hence, we are able to reduce the power without a proportional drop in utility. It is clear in Fig. 10(a) that a proportional throttling mechanism will not be able to meet the power budget without sacrificing significantly on utility.

Using Utility for Fairness The *BrownMap* power manager is able to deal with reduced power budgets by taking away resources from the lowest priority applications. This may lead to an unfair situation, which may not be acceptable to all data center administrators. We next show that utility maximization is a flexible tool that can capture diverse requirements including fairness.

In order to investigate if *BrownMap* can meet a power budget, while ensuring fairness, we change the original utility values to a more fair utility functions, namely (i) *daxpyH*: Util(ρ) = $\frac{5\rho}{33600}$, (ii) *daxpyL*: Util(ρ) = $\frac{5\rho}{33600}$, (iii) *Rubis*: Util(ρ) =$\frac{5\rho}{129}$. This ensures that for the same resource used, all applications get the same utility. We study the throughput achieved by each application in Fig. 10(b) and observe that all applications achieve the same normalized throughput in such a scenario after the reconfiguration. This study clearly establishes

the flexibility of *BrownMap* to deal with diverse optimization scenarios from strict priority to fairness.



(a) (b)

Fig. 11. Impact of migration on (a) application throughput and (b) VIO CPU Utilization

Characterization of Reconfiguration Overheads The *BrownMap* power manager performs reconfiguration actions to meet a power budget. We next investigate the impact of this reconfiguration.

Cache Contention: Earlier work has observed a drop in throughput due to hardware cache flushes during migration [25]. Initially, we observed minimal throughput drop during migration for both our applications. We conjectured that this minimal drop in throughput during migration may be because our *daxpy* application executes small jobs whereas *Rubis* has a very large memory footprint and does not use cache much. We then used a medium memory footprint version of *daxpy*, which consequently showed a throughput drop of 20% during migration (Fig. 11(a)). Our observations thus confirm that cache contention between applications with medium memory footprint can deteriorate performance.

VIO Overhead: During our experimental study, we also observed that the migration leads to an increase in CPU utilization for the VIO (Figure-11(b)). This was true for all the applications studied. This increase in VIO CPU is because of the fact that VIO has to maintain the list of dirty memory pages used by an LPAR during live migration. The increase in CPU utilization can potentially lead to an impact on performance for the LPARs during migration. However, we capture the potential virtualization overhead due to an application during the profiling step and ensure that the resource demand for a LPAR captures any CPU overhead incurred by the VIO to serve I/O requests for the LPAR. Hence, reconfiguration does not lead to a significant drop in throughput even for I/O intensive applications like Rubis.

Reconfiguration Duration: We also observed that the reconfiguration always completed in an average of 4 minutes. Further, no reconfiguration took longer than 7 minutes. The scalability of the reconfiguration is a direct consequence of the fact that we parallelize the configuration with one thread per server in the data center. This distributed reconfiguration leads to a small reconfiguration period ensuring minimal impact on applications as well as enabling *BrownMap* to scale to large shared data centers. In a large data center with thousands of servers, it may not be possible to run thousands of threads on a single node. In such a scenario, the *Reconfig Manager* can be implemented in a distributed manner on multiple nodes, where each instance of *Reconfig Manager* handles the reconfiguration for a server pool. The separation of the reconfiguration module from the placement intelligence thus allows our design to scale to cloud-sized data centers.

6 Related Work and Conclusion

Brownout is a scenario where there is a temporary reduction in power, as opposed to a complete blackout. Brownouts affect data center operations by forcing them to bring down services and applications which may seriously impact its revenue. The related work in the field of brownout management includes power minimization, virtual machine placement and power budgeting.

Power Minimization: There is a large body of work in the area of energy management for server clusters. Chen et al. [5] combine CPU scaling with application provisioning to come up with a power-aware resource allocation on servers. Chase et al. post a very general resource allocation in [4] that incorporates energy in the optimization framework. However, most of the power minimization work is in a non-virtualized setting, where short-term decisions in response to workload variations or power shortages can not be made. Other approaches to minimize energy consumption include energy-aware request redistribution in web server and usage of independent or cooperative DVFS [3, 10, 13, 14, 22].

Virtual Machine Placement: The placement of virtual machines on a server cluster has been studied in [23, 17]. The focus in [23] is on load balancing as opposed to power budgeting. In [17], the authors advocate presenting guest virtual machines with a set of soft power states such that application-specific power requirements can be integrated as inputs to the system policies, without application specificity at the virtualization-level.

Power Budgeting: There are other efforts in reducing peak power requirements at server and rack level by doing dynamic budget allocation among sub-systems [12] or blades by leveraging usage trends across collections of systems rather than a single isolated system [20]. However, the goal of this work is not to operate within a power budget but to find a peak aggregate power. In cases where the operating power exceeds the predicted peak, each server is throttled leading to a performance drop. Moreever, these techniques do not leverage virtual machine migration that allows a server to be freed up and put to a standby state. Finally, the presence of multiple virtual machines with possibly different SLAs on a single server make resource actions at server-level impractical.

In an earlier work [25, 26], we have proposed power minimization mechanisms that use virtual machine migration to minimize power. However, the work deals with power minimization as opposed to a power budgeting problem. Also, loss in utility of virtual machines is not considered. One may note that the power budgeting problem involves power minimization as a component and is a much more general problem. In this work, we address the problem of handling a brownout with minimum loss in utility and present the design and implementation of *BrownMap*, a power manager that quickly adapts to a brownout scenario by reducing the power consumption within the budget. We present both theoretical and experimental evidence to establish the efficacy of *BrownMap* in dealing with brownouts. Our evaluation on a real cluster of IBM Power6 JS-22 Blades using real production traces indicates that *BrownMap* meets power budget with a drop of 10% in overall utility for a power reduction of 50% for realistic utility models. The reconfiguration operation in *BrownMap* is completely distributed, allowing it to scale with increase in the size of data center.

References

1. Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing sla violations. In *IEEE IM*, 2007.
2. B. Carreras, D. Newman, I. Dobson, and A. Poole. Initial evidence for self-organized criticality in electric power system blackouts. In *HICSS*, 2000.
3. J. Chase and R. Doyle. Balance of Power: Energy Management for Server Clusters. Proc. HotOS, 2001.
4. J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proc. SOSP*, 2001.
5. Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *Sigmetrics*, 2005.
6. L. Cherkasova and R. Gardner. Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In *Usenix ATC*, 2005.
7. DAXPY. http://www.netlib.org/blas/daxpy.f.
8. Gregg Dixon. Demand response for today's datacenters. In *Focus Magazine.*, 2008.
9. D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full system power analysis and modeling for server environments. In *WMBS*, 2006.
10. M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *Proc. USITS*, 2003.
11. H.-Y. McCreary et al. Energyscale for ibm power6 microprocessor-based systems. In *IBM Journal for Research and Development*, 2007.
12. W. Felter, K. Rajamani, T. Keller, and C. Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *SC*, 2005.
13. T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *Proc. PPoPP*, 2005.
14. T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Trans. Comput.*, 2007.
15. Stratfor Global Intelligence. http://www.stratfor.com/analysis/global_market_brief_emerging_markets_power_shortages/, 2008.
16. R. Koller, A. Verma, and A. Neogi. Wattapp: An application-aware power meter for shared clouds. In *Proc. ICAC*, 2010.
17. Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *Proc. SOSP*, 2007.
18. Pacific Gas and Electric Company. Peak Choice. http://www.pge.com/mybusiness/energysavings/rebates/demandresponse/peakchoice/.
19. North American Energy Reliability Corporation. 2007 Disturbance Index Public. http://www.nerc.com/, Last Accessed. April, 2009.
20. P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *Proc. ISCA*, 2006.
21. Rice Univerity Bidding System. http://rubis.ow2.org/.
22. C. Rusu, A. Ferreira, C. Scordino, and A. Watson. Energy-efficient real-time heterogeneous server clusters. In *Proc. IEEE RTAS*, 2006.
23. A. Singh, M. Korupolu, and D. Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In *SC*, 2008.
24. KSEBoard. Availability Based Tariff. http://www.kseboard.com/availability_based_tariff.pdf.
25. A. Verma, P. Ahuja, and A. Neogi. pmapper: Power and migration cost aware application placement in virtualized systems. In *Middleware*, 2008.
26. A. Verma, P. Ahuja, and A. Neogi. Power-aware dynamic placement of hpc applications. In *ICS*, 2008.
27. A. Verma, G. Dasgupta, T. Nayak, P. De, and R. Kothari. Server workload analysis for power minimization using consolidation. In *Usenix ATC*, 2009.