# On Energy Management, Load Balancing and Replication

Willis Lang
Jignesh Patel
Jeffrey Naughton

Technical Report #1670

March 2010

# On Energy Management, Load Balancing and Replication

Willis Lang, Jignesh M. Patel, and Jeffrey F. Naughton
*Computer Sciences Department, University of Wisconsin – Madison; USA*
{wlang, jignesh, naughton}@cs.wisc.edu

**Abstract**—Energy consumption is a crucial and rising operational cost for data-intensive computing. In this paper we investigate some opportunities and challenges that arise in energy-aware computing in a cluster of servers running data-intensive workloads. A key insight is that in most data centers, servers are underutilized, which makes it attractive to consider powering down some servers and redistributing their load to others. Of course, powering down servers naively will render data stored only on powered down servers inaccessible. While data replication can be exploited to power down servers without losing access to data, unfortunately, care must be taken in the design of the replication and server power down schemes to avoid creating load imbalances on the remaining "live" servers. Accordingly, in this paper we study the interaction between energy management, load balancing, and replication strategies for data-intensive cluster computing. In particular, we show that Chained Declustering – a replication strategy proposed more than 20 years ago – can support very flexible energy management schemes.

## 1 Introduction

Servers consume tremendous amounts of energy. A recent report by the EPA [3] estimates that in 2006, the servers and data centers in the US alone consumed about 61B kilowatt-hours at a cost of $4.5B, which accounts for about 1.5% of the total US electricity consumption. If current methods for powering servers and data centers continue to be used, then it is predicted that this energy consumption will nearly double by 2011. Furthermore, energy costs are quickly rising as a component of the total cost of ownership (TCO) for servers. In fact, it is estimated that in 2009, the three-year cost of electricity per server will exceed the initial cost of a server [13]. This trend is likely to get worse, with the energy component of TCO going up over time, since the processor performance follows Moore's Law and doubles (in number of cores) every 18 months, while the performance per watt only doubles every two years [10]. Thus, energy is likely to continue to be a dominant cost factor in cluster computing and data center deployments.

There are many ways one can attack the problem of server energy consumption, ranging from new power distribution schemes, to new hardware architectures, to software mechanisms for energy-aware management of computation. Many of these techniques are complementary and can be used with each other. In this paper, we focus only on the software mechanisms.

One striking observation is that the servers in most data centers run at low utilization, that is, the average utilization of servers is typically in the 20-30% range [8]. If servers consumed, say, 20% of their peak load energy requirement when running at 20% utilization, this would not be a problem — the underutilization of servers would lead to an underconsumption of energy. Unfortunately, the real situation is the opposite — lightly loaded servers consume a substantial fraction of the energy they consume at peak loads. (We present results from our experiments demonstrating this phenomenon in Section 2.1.) While this is unfortunate, it does suggest the interesting possibility that one can run a given workload with less energy by powering down underutilized servers and redistributing their load to the remaining powered-on servers. However, this is problematic if the cluster is being used in a fashion where a large data set is distributed across the disks attached to the servers (as is the practice for parallel DBMSs.) In such systems, powering down servers can render a portion of the data unavailable.

Fortunately, most clusters servicing data-intensive workloads already employ data replication schemes, to ensure data availability and reliability in the presence of failures. One of our key observations is that *this same replication can be exploited to ensure availability in the presence of deliberate server power downs intended to save energy.* However, while data replication can indeed be exploited to power down servers without losing access to data, care must be taken in the design of the replication and server power down schemes to avoid creating load imbalances on the remaining "live" servers, which can have severe performance consequences.

To see this point, consider a system that uses the common replication strategy of mirroring partitioned data. To make this example more concrete, suppose that there are four nodes using mirrored replication. In addition, suppose that the data set is split into two partitions, $P_0$ with mirror $R_0$, and $P_1$ with mirror $R_1$. Assume that node $n_0$, $n_1$, $n_2$, and $n_3$ store $P_0$, $P_1$, $R_0$, and $R_1$ respectively. Furthermore, assume that queries can be sent to either the primary copy or the replica for load balancing. If the overall system utilization is at or below 50% of the provisioned utilization, then nodes $n_2$ and $n_3$ could be turned off to save energy, while nodes $n_0$ and $n_1$ would then operate at 100%

utilization. This is an ideal scenario and may be sufficient for certain systems. However, we wish to explore powering down nodes when utilization is between $50 - 100\%$ for a finer grained energy management scheme.

Now, consider another scenario in which the four nodes each initially see a load of 75%. The system has the capacity to run this workload on only three processors. Furthermore, by exploiting replication, we can certainly turn off one processor and still maintain access to all data.

Unfortunately, if we turn off node $n_3$, then nodes $n_0$ and $n_2$ will continue to operate at 75% utilization, but now both node $n_1$ and node $n_3$'s original load will be directed at node $n_1$, so the presented load there will be 150%, and the system will likely fail to meet its performance requirement. Such large load imbalances may be acceptable in certain environments, but the performance degradations are usually unacceptable (see Sections 2.1 and 3.1 for more details).

Given this example, our goal is to investigate the interaction between replication and power down schemes to provide the foundation for energy management approaches that gracefully adapt to overall system utilization. This should be done in such a way as to maximize energy efficiency by powering down some nodes while ensuring that the utilization of the remaining nodes does not exceed a targeted peak utilization.

The database and distributed systems communities have a rich history of designing various replication schemes for reliability [6], [11], [12], [21], [28], [40]. This raises the question of whether or not there is a replication scheme that can be exploited to better meet our goals than the commonly used mirroring strategy adopted in our example above. As we will demonstrate, the surprising answer is yes — one of the earliest proposed parallel database data replication schemes, the "Chained Declustering" technique [21], when coupled with careful choices of which nodes to power down, can be exploited to achieve the above goal.

In this paper, we explore node power down sequences that leverage Chained Declustering to mitigate the load imbalances created by other replication and power down sequences. We present two node power down techniques, called "Dissolving Chains" and "Blinking Chains", that view the nodes in the cluster as a "chain" and then specify which nodes are powered down as load drops (the power up sequence in response to an increasing load follows a reverse strategy, as discussed in Section 4). In Dissolving Chains, as system utilization decreases, it simply powers down more nodes (the chain dissolves). Blinking Chains differs in its power down transition because it may first power up some nodes before powering down the desired number of nodes (the chain blinks) in order to reduce load imbalances.

To the best of our knowledge this is the first paper exploring this interaction between power down sequences and replication strategies while controlling load imbalances.

In addition, we also evaluate these techniques using an extensive experimental methodology, which includes using an actual commercial DBMS, and show that: (1) given an input parameter, namely, the percentage load imbalance the power management scheme is allowed to introduce, our method guarantees that it will not introduce any additional load imbalances beyond that percentage. This percentage refers to the tolerable load imbalance that the system is allowed to take. As we will see, our methods produce low imbalances (none at some points), and this measure can be used by the system to determine if a certain power down transition is acceptable. (2) Our methods have the potential to produce significant energy savings (of $40\%$ or more) over a wide variety of system loads while maintaining data availability and a well-balanced system; and (3) our methods provide a trade off between mitigating load imbalance and ease of transitioning between operating states.

The remainder of this paper is organized as follows: Section 2 presents the problem statement. Previous replication methods are described in Section 3. Our methods are presented in Section 4, and evaluated in Section 5. Related work is discussed in Section 6; and Section 7 contains our conclusions and directions for future work.

## 2 Background and Problem Specification

Before we proceed, we define a few terms that we use throughout this paper. We use the term *load* on a node to refer to the work that is being carried out on a node. In a system with a number of concurrent queries, each with the same processing cost, the load can simply mean the number of queries per node.

The term *utilization* of a server node refers to the resource consumption on the node. Typically utilization of a system in cluster environments is measured simply as the CPU utilization [8], [17], which is a simplistic measure as it ignores other resources such as memory, disk, and network, but often works well in practice. The term *overall system utilization* refers to the average utilization across all the server nodes in the system. *Maximum node utilization* refers to the maximum utilization across all the server nodes.

Often cluster systems are designed to handle a certain provisioned *peak* load. We will often refer to the utilization using a value expressed as a percentage. Within this context, a utilization of 100% simply refers to operating at an initial designated "peak load" (which could be lower than the system's peak load at which it is stable). Lower utilization values, e.g., 50%, imply a corresponding reduction in the load (and an increase in server idle time).

The energy management schemes that we describe in this paper work by taking some nodes *offline*, which refers to a node being powered down to save energy. Nodes that are available to run queries are *online*. An offline node becomes available when it is powered up, in which case it then comes online. (In the more traditional case of replication for failure management, offline refers to the node being unavailable due to some component failure.)

Finally, an operational state for the entire system is defined as:

*Definition 2.1:* The **operating state** of the entire system, $s(m)$, is a state where $m$ of the $N$ total nodes in the system are offline.

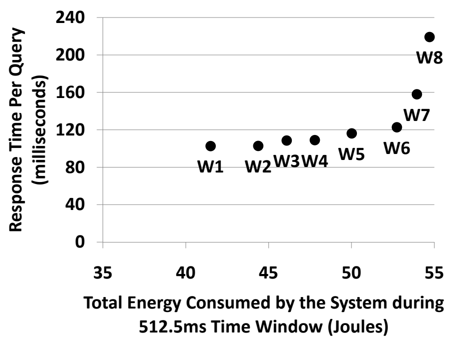


Fig. 1. Energy Consumption and Response Time Profile

## 2.1 Server Load vs. Energy Consumed

In this section we discuss the interaction between the load on a server and the energy consumed by the server. The main point here is that this relationship is not linear — even at zero load, a server consumes an unfortunately high fraction of its fully loaded energy requirement, mirroring the observation in [8], but for DBMSs.

As an example, consider Figure 1, which shows the characteristics of a 1% clustered index query workload running on a commercial DBMS. (Each point is actually an average over a thousand runs; more details about this workload are presented in Section 5.2.) In this graph, the point W1 corresponds to a server workload in which one instance of the query takes X ms to run followed by the server being idle for 4X ms. One can view this workload as a series of time windows, each of size $5X$ ms, where $X$ is the time to run the query. For workload W1, only one query is run in each window.

Other points in this graph correspond to higher server utilizations, which we achieved by randomly adding more queries in the time window (of length 5X ms), thereby reducing the idle component. Specifically, a point W$i$ corresponds to injecting $i$ queries, with random arrival times, into each $5X$ ms time window. Figure 1 shows for each workload the average execution time per query and the energy consumed by the server to run the workload.

Now, consider the point W1 in Figure 1. In this case, the server consumes about 41.5 Joules and provides a query response time of 102.5 ms. Most of this energy, specifically 74%, is consumed while the server is idle. As we add more queries to the workload, i.e., go beyond W1, the idle time decreases and a larger fraction of the energy consumed by the server is spent actually running the queries. At W5, since each query takes X ms to run, we are running at some provisioned “peak” utilization of $100\%$. Notice how performance rapidly degrades beyond W6. Operating at such points (W7 and beyond) merely to save power may be unacceptable as this region likely represents an unstable operating range.

If efficiency is defined as the energy consumed by the server per query, of the five workloads W1 to W5, W5 has the highest efficiency. Notice, however, the response time per query is slightly worse at W5 than at the other four points, since at the other points there is less contention for resources across different queries.

Thus we have two possibly conflicting optimization goals. The first is the traditional one — we could simply optimize for response time, which means running the system at point W1. However, typically in data center environments, the performance constraint to meet is not “as fast as possible;” but rather, something more like “no worse than $t$ seconds per query for this workload.” When agreeing to such Service Level Agreements (SLAs), data center service providers tend to be conservative and agree to performance that they can generally guarantee under the heaviest provisioned load, rather than performance they can meet in the best case. Consequently, the second optimization goal, and the one that we focus on in this paper, is to reduce the energy consumption while staying below a response time target.

## 2.2 Problem Statement

We want an energy management scheme that starts with an operating state $s(m)$ for a system with maximum node utilization of $u$ ($u < M$). Here $M$ refers to some maximum tolerable system utilization (perhaps defined by an SLA). We want the system to move to a new operating state $s(m')$ with maximum node utilization $u'$ such that $u' < M$ and $m \leq m'$, and at least one copy of each data item is available on the remaining servers that are still powered up.

Note that $M$ is defined relative to the initial designated peak load (see discussion at the beginning of Section 2). Consequently, $M$ can be greater than $100\%$; e.g., if the maximum tolerable response time is $120ms$ in Figure 1, then $M$ is $120\%$ (at W6).

Notice that the problem statement also allows setting M to 100%, in which case no node operates over the designated peak capacity.

In addition, in our problem formulation we require “data availability” – i.e., the power down sequence does not deliberately make any data item unavailable on the live servers that are powered up. We make this assumption since the time it takes to bring up a powered down server can be very high (e.g., booting up from system-off or from hibernation – see Section 5.5), and any queries against data that is made unavailable by a power down scheme will incur this latency. This high latency/delay may be unacceptable, and also makes it harder to maintain the fault-tolerance property of replication in the presence of updates (See Section 4.3). An interesting direction for future work is to consider relaxing this assumption. (As the reader will see, this paper presents many such twists that we hope will fuel further research in this emerging field of energy management for data processing systems.)

The schemes that we present differ in the “variance” in the load across the different nodes. In other words, some schemes result in larger variation in the loads across the nodes (cf. Section 5.4, Figure 6). While load variance (imbalances) are inevitable, and minor load imbalances do not create a problem, artificially creating major load imbalances can result in the system failing to meet its targeted performance (e.g., W7 and W8 in Figure 1). Accordingly, we require that the energy management techniques bound the load imbalances ($M$) that they introduce.

The parameter $M$ can be set based on what the system administrator feels is a comfortable upper bound for that system (e.g., W6 in Figure 1). Note such a bound is important as it provides a guarantee that the energy management method will not introduce unbounded load imbalances. We expect that there might be other sources of imbalances that the system might face, such as flash crowds. In such situations, the system can be pulled out of energy-savings mode. Now the situation is the same as what happens today when systems are faced with sudden load changes. The system can then execute whatever method it is currently using to deal with load fluctuations. It is an interesting direction of future work to see if we can improve upon this scheme to more deeply integrate flash crowd load management and prediction with energy management techniques that are proposed here, and/or to pick $M$ automatically based on other system operational settings.

Finally, for certain system states, the nodes can be "perfectly balanced" – which means that each online node has the same node load. In Section 4.2.3, we discuss these perfectly balanced states.

## 3 Replication Revisited

In parallel and distributed data processing systems, replication allows continued access to data when some nodes fail. Here we want to exploit replication for a related but different purpose: namely, allowing continued data access not when nodes fail, but when they are deliberately powered down to save energy, while controlling the resulting load imbalance.

When we look at the commonly used techniques: RAID [28], Mirrored Disk [11], [12], and Interleaved Declustering [40], we find that they all produce undesired load imbalances as nodes become inoperable or do not allow us to turn off multiple nodes. For instance, Interleaved Declustering retains load balance when one node fails but loses data availability if any additional nodes are lost.

RAID storage uses an array of disks controlled either by hardware or software to act as a single unit. Different RAID levels define different storage properties such as parallel data access, data redundancy, and data recoverability. However, RAID suffers from load imbalances when operating in failure mode. For example, in RAID 1, if a disk fails, the redundant copy disk must now handle all the requests that were shared across the two disks. Recent methods for a Power-Aware RAID [43] attempt to solve this problem with distinct energy saving operating states. However, these methods require pre-determining all the operating states and are generally not adaptable to changes in data size.

Our goal is to leverage a replication scheme to safely and easily power down any number of nodes for energy efficiency, and exploit the load balancing and failover properties of replication.

Note that we are powering down nodes to save energy, but the node has not failed. In other words, our schemes don't change the fault-tolerance property of replication (updates require special care as discussed in Section 4.3).

### 3.1 Mirroring Replication

The basic principle used in mirroring [11], [12] is to make a second copy of the data and store it on a different storage device. Mirroring can be implemented in a variety of different ways. One mechanism is to have disk pairs (RAID 1), with one disk storing the primary copy and the other storing the mirrored copy [12]. Access to the disks could have redundancy (e.g., there could be dual ports) so that if a controller fails the disk can be accessed from a different port (but this adds hardware costs). Mirroring can also increase parallelism by allowing queries to use either copy. When one disk fails, the mirrored copy takes over the work of its pair. However, this technique doubles the load on the disk that is still up.

There are a variety of different ways of mirroring data. However, in most schemes, when some disk fails, the load on the remaining copies goes up dramatically. For example, if we use a 2X replication scheme, in which we have a primary copy and one additional replica, then when a disk with either of these copies fails, all the load from the failed disk is transferred to the remaining disk. Thus, if we say that the cluster system can only operate as fast as its bottleneck, when a node is taken offline, the system operates at 2X load. In other words, if one decides to take one node offline in a mirrored scheme, from a load perspective, one might as well take half of the system offline. This means mirroring essentially has only two operating states, $100\%$ online nodes or $50\%$ online nodes. If a 2X increase in load is unacceptable (results in a maximum node utilization beyond an acceptable threshold), then with this scheme, there is no energy savings if the system load is between 50 and 100%. Our goal is to design schemes that will let us power down an appropriate number of nodes at *any* utilization between $50 - 100\%$ without creating unacceptable overloads, given a 2X replication scheme.

Fortunately, Chained Declustering [21] seems to have the properties that allow this exploration. Chained Declustering can lose multiple nodes in the cluster and maintain data availability. For this reason, in the rest of this paper, we consider techniques built upon Chained Declustering.

### 3.2 Chained Declustering (CD)

Chained Declustering [21] is a replication scheme that stripes the partitions of a data set two times across the nodes of the system, thereby doubling the amount of required disk space. The main hallmark of this scheme is its tolerance to multiple faults along the chain, if those faults do not occur on adjacent nodes. Furthermore, along with high availability, the arrangement of the replicas along the chain allows for balanced workload distribution when some nodes are offline. If one thinks of all the nodes

TABLE 1
An 8 node Chained Declustered ring without failure.

| Nodes: | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ |
|---|---|---|---|---|---|---|---|---|
| Primary: | $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ |
| Backup: | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $r_6$ | $r_7$ | $r_0$ |
| Load: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

TABLE 2
An 8 node Chained Declustered ring with 1 failure.

| Nodes: | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ |
|---|---|---|---|---|---|---|---|---|
| Primary: | — | $R_1(1)$ | $R_2(\frac{6}{7})$ | $R_3(\frac{5}{7})$ | $R_4(\frac{4}{7})$ | $R_5(\frac{3}{7})$ | $R_6(\frac{2}{7})$ | $R_7(\frac{1}{7})$ |
| Backup: | — | $r_2(\frac{1}{7})$ | $r_3(\frac{2}{7})$ | $r_4(\frac{3}{7})$ | $r_5(\frac{4}{7})$ | $r_6(\frac{5}{7})$ | $r_7(\frac{6}{7})$ | $r_0(1)$ |
| Load: | 0 | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ |

in the system as being arranged in a ring or chain, then Chained Declustering (CD) places a partition and its replica in adjacent nodes in the chain.

As an example of CD, consider a data set $R$, spread over $8$ nodes in Table 1. Here the primary copies of the data set are $R_0$ ... $R_7$. The corresponding replicas are shown as $r_0$ ... $r_7$. The nodes $n_0$ ... $n_7$ are conceptually organized in a ring. Primary copy $R_i$ is placed on node $i$ and its replica $r_i$ is placed on the “previous” node. During normal operation, if the access to all the partitions is uniform, then the queries simply access the primary partitions while updates in CD go to both partitions.

Now consider what happens when a node is taken offline by our energy management methods. Table 2 shows what happens when node $n_0$ is offline. Since node $n_0$ holds the partition $R_0$, all queries against this partition must now be serviced by node $n_7$, which holds the only other copy of this partition. But simply redirecting the queries against partition 0 to node $n_7$ could double the load on node $n_7$. CD solves this problem by redistributing the queries against partition 7 across both copies of that partition's data, namely $R_7$ and $r_7$. It does this for all the partitions, and ends up with a system in which each node is serving the same number of queries (i.e., the system is balanced after the node failure.) As shown in Table 2, for partition 7, $6/7^{th}$ of the queries are directed to node $n_6$ and $1/7^{th}$ of the queries are directed to node $n_7$. The distribution of the queries for the other partitions are shown in brackets in Table 2. The load on each node is balanced and is $8/7$ times the load on the node when all nodes were online.

While Table 2 shows what happens when one node is offline, CD can also tolerate additional nodes going offline. In fact, it can allow all failures in which two consecutive nodes are not both offline. One can easily see why this is the case in Table 1 and 2. If adjacent nodes fail, an entire partition will be lost since CD places backup partitions in adjacent nodes, which leads to the definition below.

*Definition 3.1:* A node in the Chained Declustering scheme is **essential** if removing it makes the data stored in the Chained Declustering scheme unavailable.

In fact, CD can allow up to $N/2$ alternating nodes to go offline, where $N$ is the number of nodes in the system. These $N/2$ offline nodes result in the uniform doubling of load across the remaining nodes in the system (provided $N \mod 2 = 0$). We exploit this property of CD to develop various energy management schemes.

## 4 Exploiting Replication for Energy Management

We can now design schemes to exploit CD to manage the energy consumption of a cluster system when the overall system utilization is less than the peak utilization (i.e., 100% utilization, using the terminology described in Section 2). Recall that from the discussion in Section 2 we want to control load imbalances such that we obey the constraint of the utilization parameter $M$.

While CD can tolerate a variety of configurations with nodes/servers being offline, as we show below, some of these configurations lead to system load imbalances. The protocol that is used to take nodes offline directly determines the uniformity and balance of the load on the remaining online nodes.

For the discussion below, we introduce a few additional terms: a *ring* refers to the logical ordered arrangement of all the nodes in a CD scheme. When a node in a ring goes offline, the ring is *broken* and produces a *segment*. Additional node failures partition segments into other segments. Each segment has two *end nodes*.

Now, consider the following proposition:

*Proposition 4.1:* If the **ring** or a **segment** of a Chained Declustered set of nodes is broken because a node goes offline, then the two new end nodes of the resulting segment(s) are **essential**.

The proof follows directly from the properties of CD and Definition 3.1, and is omitted here.

From Proposition 4.1 it follows that to take nodes offline any scheme must select additional nodes from the remaining online nodes that are not end points of the remaining segments.

Now we present two protocols for selecting which nodes to take offline. The Dissolving Chain scheme walks along segments of the ring, taking nodes offline at the halfway point of the given segment. In contrast, the Blinking Chain scheme spaces offline nodes on the ring evenly to achieve better load balancing.

### 4.1 Dissolving Chain (DC)

The Dissolving Chain (DC) protocol sequentially withdraws nodes using Proposition 4.1 so that data is always available. A simple generic algorithm to implement this scheme is shown in Algorithm 1. The input parameters to this Algorithm are: the current state ($s(m)$), and the number of offline nodes ($m'$) in the target state. At the end of running Algorithm 1, the system will be in the new state $s(m')$.

**Algorithm 1** Dissolving Chain

```
INPUT: $s(m)$, $m'(m' \leq N/2$ for an N node system)
$Q \Leftarrow$ Seg($s(m)$) //Extract segments in the current state.
$Q \Leftarrow$ Sort($Q$) //Sort in descending order of segment lengths.
$curr \Leftarrow$ NumOfflineNodes($s(m)$)
while $curr \neq m'$ do
    $seg = Q.pop$
    if $\mid seg \mid > 2$ AND $1 < \lceil(\mid seg \mid)/2\rceil < \mid seg \mid$ then
        $Q.push(seg_{1\ldots\lceil(|seg|)/2\rceil-1})$
        $Q.push(seg_{\lceil(|seg|)/2\rceil+1\ldots|seg|})$
        $seg_{\lceil(|seg|)/2\rceil}$.turnOff //turn off this node
        $curr \Leftarrow curr + 1$
    end if
end while
```

**Algorithm 2** Transition Controller

```
INPUT: $N$, $s(m)$, $m'$, $U$, $M$
$L \Leftarrow$ maxlen$(T, N, m')$
if $m' \leq N/2$ and $U(L+1)/L \leq M$ then CALL DissolvingChain($s(m)$, $m'$)
```

From the properties of CD, nodes in a longer segment of a CD ring have lower loads compared to a node in a shorter segment. By choosing to power down the middle node in a segment, we are both minimizing load imbalance as well as the load increase on the remaining nodes, as a result of the transition.

Continuing the example from Table 2, the second node that can be taken offline is node $n4$. The resulting system is balanced (uniform node load) as shown in Table 3.

Now that we have taken two nodes offline, let us consider taking another node offline to reduce the energy consumption in response to a lowering of the overall system utilization. Notice in Table 3 we have two segments of length 3. To take the next node offline, we can pick one of these two segments and cut it into two equal parts. However the load on the system now becomes imbalanced as illustrated in Table 4. Essentially, DC can only reach a balanced state when the number of offline nodes is $2^i$ and $2^i$ divides $N$, where $N$ is the number of nodes in the system.

The DC algorithm can solve the problem defined in Section 2.2 if it is implemented within a wrapper controller algorithm. This is because Algorithm 1 is not aware of the the maximum $M$ utilization requirement. The controller algorithm is given in Algorithm 2. This algorithm takes as input: the size of the system $N$, current operating state $s(m)$ desired number of offline nodes $m'$, the current system utilization $U$, and the utilization requirement $M$. Function **maxlen** calculates the maximum segment length that would be produced using $T()$ if this transition occured. For DC, **maxlen** first puts value $N-1$ into an empty queue. It then pops the top value in the queue ($y$) and pushes $\lceil(y-1)/2\rceil$ and $\lfloor(y-1)/2\rfloor$ into the queue. This is done $m'-1$ times and then the maximum value in the queue is returned. If

TABLE 3
Dissolving Chains at $s(2)$

| Nodes: | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ |
|---|---|---|---|---|---|---|---|---|
| Primary: | — | $R_1(1)$ | $R_2(\frac{2}{3})$ | $R_3(\frac{1}{3})$ | — | $R_5(1)$ | $R_6(\frac{2}{3})$ | $R_7(\frac{1}{3})$ |
| Backup: | — | $r_2(\frac{1}{3})$ | $r_3(\frac{2}{3})$ | $r_4(1)$ | — | $r_6(\frac{1}{3})$ | $r_7(\frac{2}{3})$ | $r_0(1)$ |
| Load: | 0 | $\frac{4}{3}$ | $\frac{4}{3}$ | $\frac{4}{3}$ | 0 | $\frac{4}{3}$ | $\frac{4}{3}$ | $\frac{4}{3}$ |

TABLE 4
Dissolving Chains at $s(3)$

| Nodes: | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ |
|---|---|---|---|---|---|---|---|---|
| Primary: | — | $R_1(1)$ | — | $R_3(1)$ | — | $R_5(1)$ | $R_6(\frac{2}{3})$ | $R_7(\frac{1}{3})$ |
| Backup: | — | $r_2(1)$ | — | $r_4(1)$ | — | $r_6(\frac{1}{3})$ | $r_7(\frac{2}{3})$ | $r_0(1)$ |
| Load: | 0 | 2 | 0 | 2 | 0 | $\frac{4}{3}$ | $\frac{4}{3}$ | $\frac{4}{3}$ |

the number of offline nodes is valid and the maximum node utilization constraint ($M$) is not violated, the controller calls the transitioning algorithm.

A companion algorithm is also required to bring nodes online when utilization increases. In a simple implementation, this companion algorithm simply reverses the transitions made by Algorithm 1. We omit this algorithm in the interest of space.

## 4.2 Blinking Chain (BC)

The general intuition behind the Blinking Chain (BC) methods is to allow more general cuts than the simple binary cuts used by DC to: a) to reduce the variation in the load across the nodes that are still up, and b) produce states where the load across the nodes is "balanced" (see Definition 4.2).

For example, for a system where $N = 40$ nodes, for a DC system at $s(9)$, there will be segments of length $4, 2, 1$ with 28 nodes at $(5/4)$ load, 2 nodes at $(3/2)$ load, and 1 node at double load. A better way to cut the $N = 40$ ring results in 4 segments of length 4 and 5 segments of length 3. This results in minimal load variation across the remaining online nodes (the benefits of this are shown in Section 5.4). We now discuss how to create these segments.

### *4.2.1 Segments and Transitions*

The general algorithm for transitions from a balanced state with $m$ nodes offline to a target state with $m'$ nodes offline is shown in Algorithm 3. The goal of this algorithm is to power down nodes such that the remaining online segments have lengths as uniform as possible. In this algorithm, one node in the ring is always deemed the *root* node. The algorithm iterates through every node in the ring starting from the root and changes the state of the node if appropriate. Notice that this algorithm can be used to both transition down (i.e., $m < m'$) or transition up (i.e., $m > m'$). After running this algorithm, the system moves to the state $s(m')$.

In Algorithm 3, when $m'|N$ (| is the "divides" operator), all the resulting segments will be of equal length. However

**Algorithm 3** Blinking Chain

```
**INPUT:** $N$,$m'$ ($m' \leq N/2$),$root$
$curr \Leftarrow root$; $s \Leftarrow 0$
**if** $m' > 0$ **then** $curr$.*turnOff*
$curr \Leftarrow root.next$; $tgtlen \Leftarrow \lceil (N - m')/m' \rceil$; $ctr \Leftarrow 0$
**while** $curr \neq root$ **do**
    **if** $ctr \neq tgtlen$ **and** $curr$.*isOff* **then** $curr$.*turnOn*
    **if** $ctr = tgtlen$ **and** $curr$.*isOn* **then** $curr$.*turnOff*
    $ctr \Leftarrow (ctr + 1)\%(tgtlen + 1)$
    $curr \Leftarrow curr.next$; $s \Leftarrow s + 1$
    **if** $s = N \bmod m'$ **then** $tgtlen \Leftarrow \lfloor (N - m')/m' \rfloor$
**end while**
```

if $N$ is not divisible by $m'$, ($N \bmod m'$) segments will have lengths $\lceil tgtlen \rceil$ and the remaining $m' - (N \bmod m')$ segments of length $\lfloor tgtlen \rfloor$.

BC can also adhere to the maximum node utilization constraint $M$ defined in Section 2.2 by using a controller algorithm similar to Algorithm 2 except that a) we call Algorithm 3 (Blinking Chain) instead of Dissolving Chain in the last line and b) **maxlen** calculates $\lceil tgtlen \rceil$ as described above.

Now, consider transitioning from a state with $m$ nodes offline to $m'$ nodes offline. A method to implement this transition is to bring all but the root node back online and then turn $m' - 1$ of them off, but this results in a high transitioning cost as each transition requires making $m + m' - 2$ node state changes (i.e., changing the state of a node from offline to online, or vice versa). These state changes can consume a significant amount of energy (see Section 5.5), and we would also *like to minimize the energy spent in making these transitions*. An interesting property of BC is that when transitioning from state $s(m)$ to $s(m')$, there may be offline nodes in the $s(m)$ configuration that can remain offline in the $s(m')$ configuration. By not changing the status of these nodes, the transitions can be made more energy efficient, as discussed next.

### *4.2.2 Optimizing the Transitions*

First consider finding states that provide the most "efficient" transitions, which implies making the least number of node state changes in the transition. In BC, the most efficient transition between two states $s(m)$ and $s(m')$ is such that only $|m - m'|$ nodes undergo transition. This efficient transition is defined formally as:

*Definition 4.1:* The **Optimal Blinking Chain Transition** $s(m)$ to $s(m')$ only requires $|m - m'|$ nodes to undergo transition.

This optimal transition can be implemented by using Algorithm 3. We now give Proposition 4.2 which highlights a key relationship between divisible states ($s(m)$, $s(m')$ such that $m|m'$ or $m'|m$) and the Optimal Blinking Chain Transition.

*Proposition 4.2:* $s(m)$ to $s(m')$ is an optimal Blinking Chain transition iff ($m|m'$ OR $m'|m$)

*Proof:* First, if $s(m)$ to $s(m')$ is an optimal transition, then only $|m - m'|$ nodes have changed state. Given any $s(m)$, we know there are ($N \bmod m$) segments of length $\lceil (N - m)/m \rceil$ and $[m - (N \bmod m)]$ of length $\lfloor (N - m)/m \rfloor$. Also, we know that there are $m$ total segments in $s(m)$. Without loss of generality, assume $m' > m$, which means exactly $x = (m' - m)$ nodes have powered down. We can prove $m|x$ by contradiction. Assume that $x$ is not a multiple of $m$, then segments in $s(m)$ are not all cut the same number of times. This means that the maximum difference between two segments in $s(m')$ cannot be 1, which is a contradiction of the above. Next, we need to prove that given $m|m'$, then the segments in $s(m)$ can be cut into the segments in $s(m')$ with $m' - m$ offline nodes. Since $m|m'$, then each segment in $s(m)$ will receive $c = (m' - m)/m$ cuts. So the lengths of the segments in $s(m')$ will be $(\lceil (N - m)/m \rceil - c)/(1 + c) = \lceil (N - m')/m' \rceil$ and $(\lfloor (N - m)/m \rfloor - c)/(1 + c) = \lfloor (N - m')/m' \rfloor$. Lastly, $x\lceil (N - m')/m' \rceil + y\lfloor (N - m')/m' \rfloor = N - m'$ and certainly ($x = N \bmod m'$) and ($y = m' - (N \bmod m')$) hold. □

Proposition 4.2 tells us that in a given operating state, $s(m)$, for $N$ CD nodes, we can transition to another $s(m')$ with maximum efficiency if and only if $m'$ is a multiple or factor of $m$. While the Optimal Blinking Chain Transition has interesting properties, it does not handle all possible state transitions. Specifically, it does not cover transitions between any states $s(m)$ and $s(m')$ when $m$ and $m'$ do not divide each other. For example, if $N = 42$, we cannot execute $s(6)$ to $s(15)$, since the optimal transition is not defined in this case.

To handle transitions between any two arbitrary states, we need a **General Blinking Chain Transition**. This transition is implemented as a composition of two Optimal Blinking Chain Transitions: $s(m)$ to $s(GCD(m, m'))$ to $s(m')$, which maximizes the number of offline nodes that are untouched during the transition.

Using our previous example, if $N = 42$ and we wish to transition from $s(6)$ to $s(15)$, then using the General Blinking Chain Transition, we can save $4$ node transitions by doing two optimal transitions: one from $s(6)$ to $s(3)$ and the second from $s(3)$ to $s(15)$. Finally, we note that since the Optimal Blinking Chain Transition can be implemented with Algorithm 3, the General Blinking Chain Transition can simply be implemented using two iterations of Algorithm 3.

Notice that BC transitions are "optimal", when only $|m - m'|$ nodes transition. Recall this is *always* the case for DC transitions. The implication of this property is discussed in Section 5.5.

### *4.2.3 Number of Balanced States*

Let us now consider the special states $s(m)$ where $m|N$. In these states, all the nodes have identical loads and we deem this "balanced" as defined in Definition 4.2.

*Definition 4.2:* If all segments of a Chained Declustered ring are of equal length ($m|N$) in a given operating state $s(m)$, then we deem this a **balanced** operating state, $\bar{s}(m)$ and all nodes have the same load.

We can calculate the total number of possible balanced operating states for a Chained Declustered system of $N$

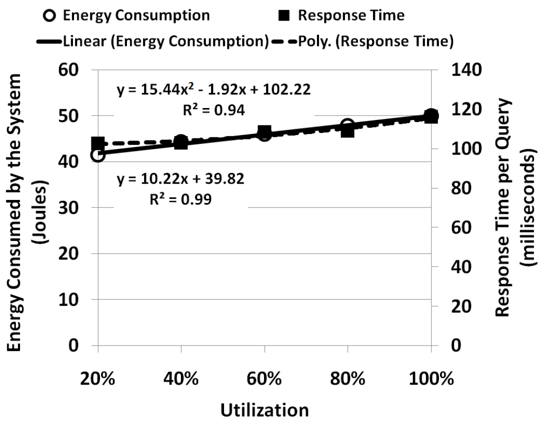


Fig. 2. Index query regression model

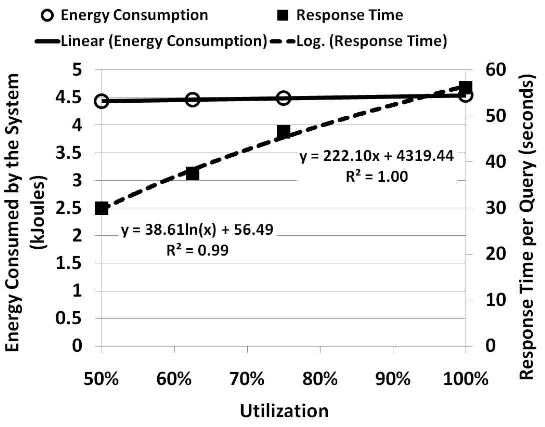


Fig. 3. Database scan regression model

nodes as follows: consider the system configured with $N = p_1^{N_1} p_2^{N_2} ... p_j^{N_j}$ where $p_i$ is the $i^{th}$ prime; by simple combinatorics, the total number of unique factors of $N$ is $\Pi_{1 \leq i \leq j}(N_i + 1)$, which is also the number of balanced states for this system since $\bar{s}(0)$ replaces factor $N$.

### 4.3 Updates

Updates while operating in energy saving modes can be handled (without sacrificing the fault-tolerance properties of replication) as follows: if an update needs to be applied to a partition replicate that is offline, then the "next left node" can store the updates applied to a partition that has a replicate powered down.

For example, consider the node segment "A–B–C–D" in a CD ring with C has been powered down. In this case, node B can store the updates that have been applied to node D, and node A can store the updates that have been applied to node B. Updates for node A go to node B as usual. When node C comes back online, the update logs stored on A and B will be applied to the partitions on node C. Note that the original fault tolerance property of replication is maintained for CD even when we are operating in energy efficient modes, as the system always keeps two copies of each update.

Section 5.6 presents results on the cost of log replay with respect to the amount of data updated and node power up costs.

## 5 Evaluation

In this section we present results evaluating the effectiveness of our energy management methods. At a high level our methodology was the following: we took an actual server and ran two prototypical workloads on the server. We then took actual measurements for both energy and response time on this server, as we varied the load on the server (i.e., changed the server utilization). We then produced a model for a single node in a system. This model was then plugged into a larger model for the entire distributed system. Using this method, we were able to explore a range of system configurations.

In all results presented below, we consider a system with 1000 nodes (i.e., $N = 1000$). Additional results with different values of $N$ are similar to those presented here, and hence are omitted.

### 5.1 Experimental Setup

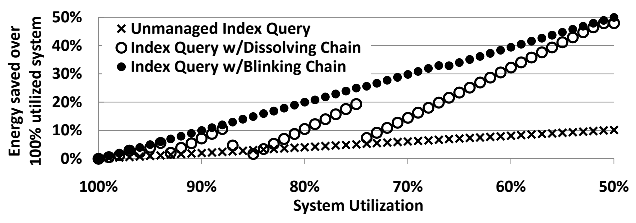
Our system under test (SUT) consisted of an ASUS P5Q3 Deluxe WIFI-AP motherboard with an Intel Core2Duo E8500, 2GB Kingston DDR3 memory, an ASUS GeForce 8400GS 256M graphics card, and a Western Digital Caviar SE16 320G SATA disk. The power supply unit was a Corsair VX450W PSU. System energy draw was measured using a Yokogawa WT210 unit as suggested by the SPEC power benchmarks [5]. The WT210 measurements were collected by a separate system through the RS232 interface and the provided Yokogawa software.

We used both a DBMS index query workload and a table scan workload (described below). The DBMS workload was run on a commercial DBMS. Our database consisted of the Wisconsin Benchmark (WB) tables [16]. Client applications accessing the database were written in Java 1.6 using the JDBC connection drivers for the commercial DBMS.

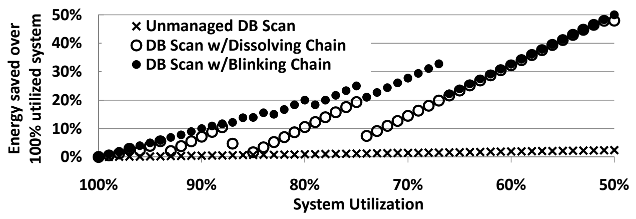
All empirical results were the average of the middle three results of five runs. The offline mode used was the hibernation (ACPI S4) state. Alternative offline modes are discussed in Section 5.5.

### 5.2 Workload

We model two different types of workloads. The first workload uses WB Query 3. This query is a 1% selection query using a clustered index. The target table for this query is a table with $20M$ tuples (approx. 4GB table size). The actual workload consists of 1000 such queries with randomly selected ranges. This workload is used to model simple lookup queries. Our second workload is a file scan on a WB table (of varying sizes) that has no indices. This workload mimics queries that require scanning tables in a DSS environment. These workloads are described in more detail below.



(a) Index Queries, N=1000



(b) Scan Queries, N=1000

Fig. 4. Energy savings under varying system utilization.

#### 5.2.1 *Index Queries Workload*

To simulate varying node underutilization with the indexed range query, we defined various workloads for the indexed query by varying idle times (this is the same setup as described in Section 2.1). First, we ran this query and measured the query runtime. Lets call this X seconds. Then, we defined a 20% utilization workload as one in which the query runs for X seconds followed by an idle time of 4X seconds. In this setup, the server is presented with a series of these 5X time windows. An actual run consists of 1000 such windows, with random arrival time for the query in each window. We average the results over each run. Workloads with higher utilization are generated by injecting additional queries in this 5X window. For example a workload with 40% utilization has two queries in each 5X window, and a workload with 100% utilization has 5 queries in each 5X window.

To determine the value of X above, we ran 10000 random 1% selection queries and measured the average response time at 102.5 ms, with a standard deviation of 0.46 ms.

#### 5.2.2 *Database Scan Workload*

We modeled utilization of the system running scan workloads slightly differently to mimic a scenario in which a single scan runs across all the nodes in the system. In this case, when nodes are taken offline, the remaining online nodes have to scan larger portions of the data. In this model, let the time it takes a node to scan a $20M$ tuple WB table be $56.49$ seconds. This node is operating at $100\%$ utilization, scanning as much as possible. For $75\%$ utilization, we ask the node to scan a $15M$ tuple table every $56.49s$. Thus, over time, it is doing $75\%$ of the work that it would do in the $100\%$ case. Similarly, for $50\%$ utilization, we ask it to scan a $10M$ tuple WB table every $56.49s$. Energy consumption is measured for the entire $56.49s$ window. With increased utilization, the increase in response time increases (nearly) linearly. All scans are “cold” and there is no caching between successive scans.

### 5.3 Modeling Energy and Response Time

In this section we present the measured energy consumption and response time results for each workload. We then use these results to develop a model for the behavior of a node in the system. All models were picked by trying a number of different linear and polynomial regression models, and picking the one with the lowest coefficient of determination, $R^2$. All presented models had $R^2 > 0.94$.

#### 5.3.1 *Indexed Query Workload*

The response time and energy measurement results for the index workload are presented in Figure 1 (in Section 2.1). Figure 2 plots this data with utilization on the x-axis, system energy consumption (for a 5X window) on the primary y-axis, and the query response time in milliseconds on the secondary y-axis.

Figure 2 also show the derived regression models for the average energy consumed by our SUT and the average query response time as a function of utilization. The energy consumption model is linear while the response time model is quadratic.

#### 5.3.2 *Database Scan Workload*

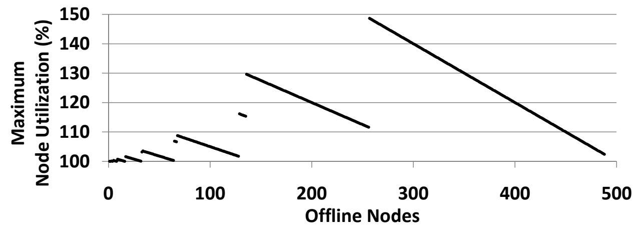
For the scan workload, increased utilization corresponds to increasing the length of time that an instance runs (to mimic what would happen if we turned nodes offline for such workloads). The results for this workload are presented in Figure 3. Again, the energy model is linear, but for scan the response time model is logarithmic. The average response time curve is sublinear as the pre-fetching used by the DBMS decreases the per-record response time as we increase the amount of data that is read. While the energy consumption curves in Figures 2 and 3 are both linear, as utilization increases, energy consumption grows faster with the CPU-bound index workload.

### 5.4 Effect of Decreasing Utilization

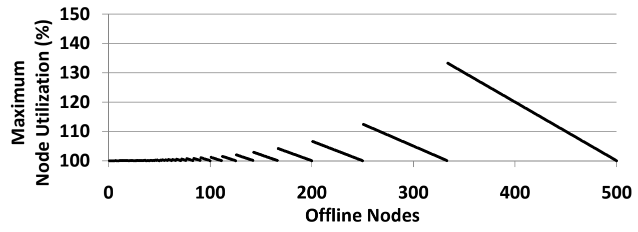
Using the models described in the previous section, we now apply the workload models to a $N = 1000$ system configuration under varied system utilization.

We then analyze the workload energy consumption of the overall system as the overall system utilization decreases from 100%. In addition to comparing differences between our methods, we also compare against the *Unmanaged* system, where all nodes are always online regardless of the overall system utilization.

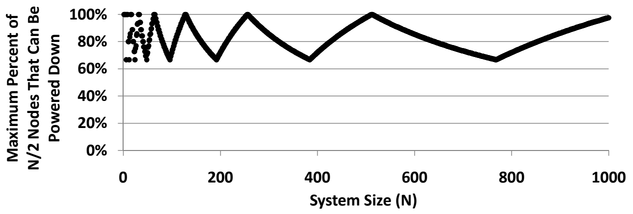
These results are shown in Figures 4 (a) and (b). In these figures, we vary the system utilization from 100% to 50% as shown on the x-axis (going from 100% on the left to 50% on the right). So going left to right, corresponds to decreasing the overall system utilization from the fully loaded (100%) system. For each point in these figures, we apply our empirically derived models from Section 5.3 to calculate the energy consumption. Using this calculated energy consumption, we plotted, on the y-axis, the energy saved by the entire system compared to the energy consumption at the 100% point.



(a) Dissolving Chains

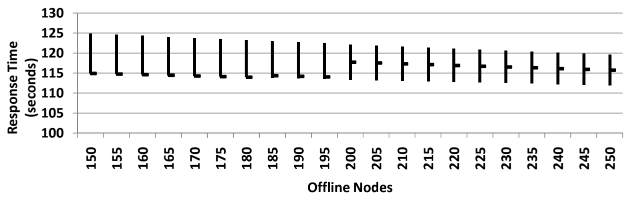


(b) Blinking Chains

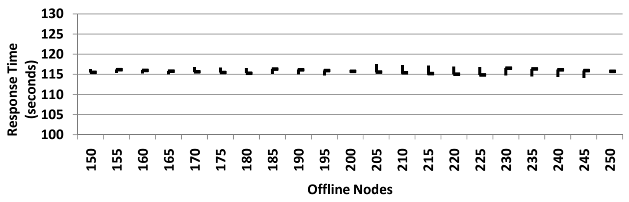


(c) Dissolving Chain maximum number off offline nodes

Fig. 5. (a-b) Maximum node utilization as we iteratively take nodes offline. (c) Ability of Dissolving Chains to power down half of the nodes.



(a) Dissolving Chains Response Time



(b) Blinking Chains Response Time

Fig. 6. Comparing imbalanced operating points using the Index Query workload. The vertical lines in represent the range between the minimum and the maximum response times and the horizontal bar is the median response time.

We notice that an unmanaged cluster saves at most $10\%$ in energy consumption (for the Index query workload Figure 4 (a)) at $50\%$ utilization. For the Scan workload (Figure 4 (b)), the unmanaged cluster only saves 3% of energy at $50\%$ utilization! However, using DC and BC, we can save 48% and 50% of the energy consumption at $50\%$ utilization respectively. Notice, because of DC's inability to power down $500$ nodes for $N = 1000$, its savings is slightly lower than BC.

Another striking observation from Figure 4 is that the curves for both BC and DC have big swings/spikes. These spikes can be seen for both methods clearly in Figure 4 (b). This behavior is because both methods introduce load imbalances at certain operating states. Notice that the swings for BC are more gradual compared to DC – this is because BC maintains optimal load balance on the online nodes at any given operating state, which makes its energy swings are more subtle compared to DC.

Let us explore these swings in greater detail. Consider Figures 5 (a) and (b), where we power down $m$ nodes when the system utilization is $(1000 - m)/1000$ for a $1000$ node system. As the system utilization drops, consider taking nodes offline one by one (incrementing $m$ by 1), up to a maximum of $500$ nodes, using both DC and BC methods. Note that not all states will be balanced.

Figures 5 (a) and (b) show the *maximum node utilization* for both methods, i.e., the maximum relative increase (compared to $m = 0$) that any system node will see. Note the maximum node utilization is a crude way to determine the imbalance of the system. (This type of analysis can be used to avoid load spikes seen in Figure 1.) Comparing these two figures, we see that BC is more graceful in its worst-case node utilization in imbalanced states (where maximum node utilization is greater than $100\%$) compared to DC.

In addition, from Sections 4.1 and 4.2.3 we know that BC has $16$ balanced states (see Section 4.2.3) for $N = 1000$ while DC only has $4$. (These correspond to a 100% maximum node utilization in Figures 5 (a) and (b).) Furthermore, even when both methods are imbalanced, BC has a better worst-case behavior than DC, as is evidenced by the lower height (node utilization) of the operating points in Figures 5 (a) and (b). For example, with respect to our problem statement in Section 2.2, if $M = 120\%$, then BC has $67$ states where the maximum node utilization violates this constraint while DC has $209$ states. This is simply a count of all possible operating states with a maximum nodes utilization greater than $M$.

Lastly, we notice from in Figure 5 (a) that DC cannot reach $\bar{s}(500)$ for $N = 1000$. This is because as it systematically traverses the ring, cutting segments in half, it may create irreducible segments of length $2$. Thus, it cannot reach the optimal number of offline nodes. This effect can be seen in Figure 4, where near $50\%$ utilization, DC is slightly lower in energy savings than BC.

An analysis of this phenomenon over varying system sizes ($N$) is shown in Figure 5 (c). Here we show how close DC can come to powering down $N/2$ nodes for $1 \leq N \leq 1000$. What we notice is that there are dramatic swings, but more importantly, we notice that DC can

TABLE 5
Costs for different types of offline states.

"Down" means going from the idle online state ($75.2W$) to an offline state, and "Up" is the reverse. The ASUS offline state is a proprietary idle state provided by the motherboard software.

| | Down time (s) | Down cost (J) | Up time (s) | Up cost (J) | State cost (W) |
|---|---|---|---|---|---|
| ASUS | 0.8 | 83.5 | 0.7 | 68.4 | 72.3 |
| Standby | 12.1 | 1033.2 | 14.3 | 1299.8 | 11.6 |
| Hibernate | 12.2 | 1107.6 | 37.3 | 3531.6 | 0 |
| Shutdown/Off | 8.7 | 700.2 | 177.6 | 9655.9 | 0 |

transition to $\bar{s}(N/2)$ only when $N = 2^i$. Ultimately, the reason this occurs is because DC heuristically takes nodes down and will never self-correct by bringing them back online as utilization monotonically decreases. The upside to this heuristic is a low (constant) transitioning energy cost that is discussed in Section 5.5.

For a detailed look at further effects of BC optimal load balancing to DC heuristic balancing, we zoom in on a smaller set of operating states. We use the models of Figures 2 and 3 and compare how energy consumption and response time are affected by these imbalanced states. Figure 6 examines the imbalanced operating points for the range of $150$ to $250$ offline nodes, in $5$ node increments, while executing the Index query workload (Figure 2). (The results for the scan query workloads are similar and omitted here.) Figures 6 (a) and (b) compare the variance in node response time between the operating states for DC and BC, respectively. The response time variance is clearly far smaller with BC.

To summarize, BC transitioning results in more balanced node loads than DC. With its lower maximum node utilizations, BC offers greater opportunities to power down nodes and stay within the threshold $M$ in our problem statement (see Section 2.2).

## 5.5 Effect of Transitioning Costs

So far we have not included any energy or latency costs associated with making transitions from one state to the next. There are a number of possible offline "power states" for a node. For our test system (SUT), Table 5 shows the different offline and online transitions, along with the time it takes to make the transition and the energy consumed in making the transition. To put the energy costs into perspective, our SUT has an idle power consumption of $75.2W$. While the "ASUS" state has the fastest transitioning time, it consumes $95\%$ of the idle cost, making it of limited use. The standby mode is more efficient, consuming 11.6W while keeping keeping memory and system state online. The hibernate state has no sustained cost in the offline mode, and provides faster transitions than turning the machine off. Here we use hibernation as our power down mechanism (note that machines can be powered up/down using IPMI which is fairly ubiquitous on modern servers).

TABLE 6
Example transitioning sequence, energy costs, and $\tau$

| Util (%) | BC (kJ) | DC (kJ) | $\tau$ (secs) | Util (%) | BC (kJ) | DC (kJ) | $\tau$ (secs) |
|---|---|---|---|---|---|---|---|
| 100 to 90 | 111 | 111 | 0 | 50 to 60 | 1,745 | 353 | DNE |
| 90 to 80 | 111 | 111 | 0 | 60 to 70 | 1,281 | 353 | 168.9 |
| 80 to 70 | 575 | 111 | 84.4 | 70 to 80 | 817 | 353 | 141.6 |
| 70 to 60 | 1,039 | 111 | DNE | 80 to 90 | 353 | 353 | 0 |
| 60 to 50 | 1,503 | 111 | 321.9 | 90 to 100 | 353 | 353 | 0 |

From Section 5.4, we know that BC is optimal in balancing the load across the nodes, but the cost of this optimality is a complex transitioning mechanism (cf. Section 4.2.2). In contrast, DC always powers up/down the minimal number of nodes required to reach the target operating state. (As discussed in Section 5.4, not all BC operating states are available to DC. To facilitate direct comparison, here we compare operating states that are accessible to both DC and BC). From the perspectives of energy consumed during the actual transitions, DC is clearly more efficient. Now we answer the question: *How much worse is the transition cost of BC?*

Let $O_s(N, y, z, t)$ be the energy cost of running a workload on an $N$ node system with $y$ offline nodes at $z\%$ utilization for $t$ seconds using scheme $s$ (e.g., $O_D$ and $O_B$ for Dissolving and Blinking Chains respectively).

$$\Delta O(N, a, b, z, t) = O_D(N, a, z, t) - O_B(N, b, z, t) \quad (1)$$

$$\forall y, z, t : O_B(N, y, z, t) \leq O_D(N, y, z, t) \quad (2)$$

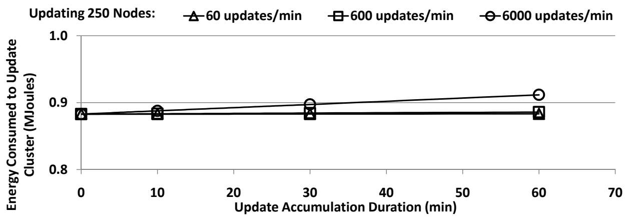
Given Equation 1 and Equation 2 (the load balancing of DC is lower bounded by BC), $\Delta O(N, y, y, z, t) \geq 0$. Given this, if $a \leq b$, the function $\Delta O(N, a, b, z, t)$ grows monotonically as $t$ increases.

$$\Delta O(N, y, y, z, \tau) > \gamma \quad (3)$$

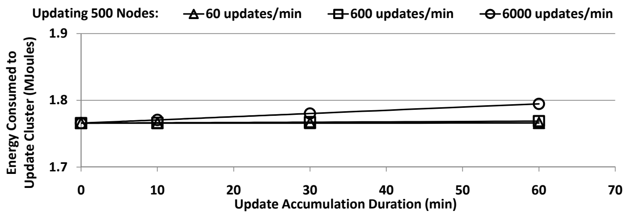
Equation 3 introduces the notion of $\tau$, which is the length of time that the two systems, one using DC and the other using BC, must operate for for the BC system to overcome its extra (with respect to DC) transitioning cost penalty ($\gamma$) with its efficient load balancing (energy efficiency).

For example, given $N = 1000$ and the Index Query workload, if we want to transition $s(200)$ to $s(300)$, BC pays $\gamma = 464kJ$ in extra transitioning cost over DC. If we have two systems where both DC and BC making this transition, then the BC system must stay at $s(300)$ for at least $\tau = 36.4sec$ to overcome this penalty, otherwise DC is a more cost (energy) effective solution.

In Table 6, we provide two example transitioning sequences for two $1000$ node systems: one using the DC scheme and the other using the BC scheme. We show a transitioning sequence (in columns 1-4) where the utilization starts at $100\%$ and falls by $10\%$ increments until $50\%$. We also show a transitioning sequence (in columns 5-8) where utilization increases from $50 - 100\%$ in $10\%$ increments. In both these scenarios, we assume that there is no time



(a) Powering up 250 nodes.



(b) Powering up 500 nodes.

Fig. 7. Energy cost of powering up nodes and updating the data partitions, given different rates of incoming updates and duration for which the nodes were powered down.

spent for the change in utilization, the decision to transition, and the actual power down sequences. That is, if there is a change in utilization, the decision to transition and the execution of the transition occur instantly.

The rows show the transitioning energy cost and the time it takes ($\tau$) for a more balanced BC derived operating state to overcome its heavy transitioning cost. We notice a number of key points. First, there are a number of transitions where $\tau = 0$ such as transitioning from 100% to 90% (i.e. $s(1000)$ to $s(900)$). In this case, both Blinking and Dissolving Chains perform optimal $|m - m'|$ node transitions and so the node transition costs are the same. Second, if both Blinking and Dissolving Chains result in states with idential load imbalance, then Blinking Chains can never overcome its disadvantage in transition cost and $\tau$ does not exist (DNE). This is seen in transitions $50\%$ to $60\%$ utilization and $60\%$ to $70\%$.

### 5.6 Update Costs

The idea of transitioning costs is extended when we consider the costs of updates. Recall from Section 4.3 that when both replicas are online the updates are applied to both replicas, but if one of the replica is offline, then the update is applied to the online copy and a log of the update is stored in the “left” node. This log is then applied when the node with the (stale) replica is powered up (in a single update transaction). In this section, we provide an analysis of the energy cost of powering up a cluster and applying the update logs that have accumulated over varying amounts of time.

Consider a 1000 node cluster powered down to 75% and 50%. Let us assume 3 different update rates of 1, 10, 100 updates per second and different periods of time 10, 30, and 60 minutes during which the updates are accumulating.

| Methods | Properties | |
|---|---|---|
| | Load Balancing | Transitioning Overhead |
| Blinking Chains | Good | High |
| Dissolving Chains | Fair | Low |
| Mirroring | Poor | Low |

Fig. 8. Comparison of energy management methods

So the powered down nodes have been down for this time and updates are accumulating, while a second log of the updates is stored on the next left node for fault tolerance (see Section 4.3). Further, let us assume that updates are uniformly distributed across the file and thus the nodes in the cluster. This means that the total number of updates that must be replayed is amortized over the number of nodes that are brought up. (We have run different variants of this setup by varying the # nodes, start and end states, update rates, and down time, and the results are similar to the ones presented here.)

Using our 20 million tuple Wisconsin Benchmark table with a non-clustered index, our SUT can update individual tuples at 933 updates per second at a cost of 0.08 Joules/update. In Figures 7(a) and (b), we show the energy cost of powering up 250 nodes and 500 nodes with the varying update rates and accumulation lengths. The cost in energy is primarily dominated by the energy spent in bringing nodes out of hibernation. We notice that in both cases, updates cause at most a 2% increase in the transitioning energy.

Finally, the time spent bringing nodes online is also largely dominated by the time spent powering up from hibernation. In the case of Figure 7(a) where 6000 queries per minute accumulates for 60 minutes, the entire update process takes 38.79 seconds, of which only 1.5 seconds is the actual time to run the update transaction while the remaining 37.29 seconds is used to bring the node out of hibernation (Table 5).

### 5.7 Summary

Now we summarize some of the practical implications of our work. In a setting where load balance is not as important, as we discussed in Section 3.1, simple mirroring can be used. The power down scheme is simple (turn off one of the two replicate nodes, causing a 2X load increase on the remaining node) and it affords the $100\%$ and $50\%$ online balanced states. However, in cases where the huge 2X load imbalances must be avoided (in most cases involving SLAs), we suggest the Dissolving Chain (DC) and the Blinking Chain (BC) methods.

The differences between DC and BC are summarized in Figure 8. If avoiding load imbalances and the variation in loads across the nodes is important, then BC offers excellent load balancing in energy saving states. However, BC requires significant state transitioning overhead that would be amplified when system utilization is highly variable. Thus, if one knows the system utilization will be highly variable, DC offers low transitioning cost but incurs slight but predictable load imbalances and offers fewer state transitions.

Finally, notice that since both schemes leverage Chained Declustering, the usage of one over the other is not exclusive; if utilization fluctuates, we can switch to DC, and if there is little fluctuation, we can switch to BC. We will examine such hybrid approaches as part of future work.

## 6 Related Work

The problem of increasing energy consumption in large-scale data processing environment has received considerable attention since the beginning of this decade, especially in the context of data center construction and operation. The increasing attention in part is driven by the minimization of the TCO for data centers [27]. Examples of efficiency methods include reducing the number of power conversions, bringing in higher voltage closer to the rack, using more efficient power supply parts, raising data center temperatures, shorter control of airflow (e.g., avoiding pumping cool air from a cooling source that is far way from the target), using a cooling tower rather than A/C, using lower performance equipment, etc. [4], [18], [19], [24], [25], [29], [37]. All these efforts have resulted in dramatic improvements in the energy efficiency of data centers, and can largely be used orthogonally to software methods that reduce energy consumption.

On the software systems side, a desired property is energy proportionality. That is, an X% utilized server should consume X% of the power that it would consume when it is at 100% utilization (peak power). One of the hurdles in achieving this behavior is the problem that idle machines typically consume a significant amount (50%) of its peak power [8]. Poor energy proportionality is caused by all the major components of the server. Certain components, such as CPU, are already efficient [9] and components such as disk are under high scrutiny [1], [14], [38]. However, software systems must also be aware of hardware capabilities or adapt its usage of hardware to also achieve energy proportional computing. One example is the Tickless kernel project which aims to change the way OS kernels operate at idle [39]. The systems community has since begun to develop energy based metrics of efficiency that place power optimization as a first-order goal [5], [36]. The Joulesort benchmark develops a sort benchmark that focuses on the energy consumed [35]. Recent work has examined how direct CPU power control mechanisms can effect energy savings and workload response time [22].

Rajamani and Lefurgy studied the effect of shutting down servers to save energy [33] and achieve energy proportionality. However, their study focuses on front end web servers where back-end database servers were left unmanaged and always on. Notice that turning off database servers is a harder problem as one has to account for data availability. Pinheiro et al. showed that turning cluster nodes on and off when node load is low can save energy [31]. They studied real web server workloads as well as a distributed Linux cluster that ran synthetic workloads of CPU and I/O benchmarks. Again, since the workloads studied web request management and application migration, there was no account of data availability. Additional methods [30], [32] either rely on learning request skew, specialized hardware, and data migration and do not explore load imbalances caused by powering down disks.

One mechanism to deal with data intensive services being powered down is to use a virtual machine (VM) solution [2] whereby we run such services on replicated VMs. Indeed this idea is gaining momentum as it is an ideal candidate for non-data intensive service migration [7], [15], [34], [41], [42]. However, using VMs when running data intensive services for the purpose of migration and equipment power down is challenging for a number of reasons: (1) the performance penalty of running data intensive services on VMs may not be tolerable in an SLA environment; (2) achieving homogeneous performance from identical VMs running on differing underlying hardware platforms is still an open problem [26]; (3) migrating VMs that serve as data nodes to achieve energy efficiency is costly in performance (migrating gigabytes of data over the network), energy (network traffic), and space overhead (disks must be tremendously overprovisioned to allow the on-the-fly replication of data). Our work is distinct as we focus on leveraging replication to reduce energy consumption, while maintaining data availability and reducing load imbalances.

Weddle et al. [43] described a RAID-based system to turn off disks to save power when utilization is low. However, their work only focuses on the disk subsystem, and not entire nodes. Furthermore, their system requires pre-setting well defined “gears”, one for each operating point for the system with some disks offline. This scheme can produce up to k-replicas of some data items for k-different operating points. If gears are not setup, then the system requires on-the-fly replication as disks are taken offline, which increases the costs of taking disks offline. Such hardware/device driver-based approaches only provide a partial solution and can be complementary to our methods (such as, using this RAID-based scheme if a RAID is present at each node). Harizopoulos et al. [20] echo this connection, and also outline some broad goals for energy-aware DBMSs.

Recent work by Leverich powers down MapReduce cluster nodes but does not consider load balancing [23].

None of these previous works have considered the problem that we address – namely, energy management using replication to maintain data availability, while maintaining a well-balanced system.

## 7 Conclusions and Future Work

In this paper we have presented energy management methods that can be used in distributed data processing environments to reduce energy consumption. We leverage the properties of replication schemes and design techniques that can take nodes offline to conserve energy when the system utilization is low. Our results show that by simply choosing an appropriate replication scheme and power down strategy, significant energy savings ($35\%$ or more in some cases) can be gained over unmanaged systems without extra hardware or data migration. Further, our methods trade off load balancing against energy efficient state transitioning, allowing the user to choose a suitable strategy. To the best of our knowledge, this is the first paper that makes a connection between replication, energy management and load balancing.

This paper seeds a number of directions for future work. First, our methods used a 2X replication, and does not exploit utilization below 50% very effectively. While our methods can be used below the 50% utilization point, they do not produce any direct additional benefits (they might get some indirect energy savings benefits from the hardware as nodes are not running near 100% utilization, e.g., point W2 in Figure 1 has a lower energy consumption compared to point W5). One direction for future work is to build on the ideas proposed in this paper and broaden the connections between generic levels of replication and energy management. Other directions for future work include incorporating workload modeling and prediction techniques to work with our method, techniques that switch between Blinking and Dissolving Chains based on hybrid workload characteristics, and improving the techniques for handling rapid transitions between different operating states.

Finally, we fully recognize that replication, power down sequences, and load balancing are only part of a larger software solution for energy management in data intensive computing environments. We recognize that extensions to our work are needed to produce fully deployable complete solutions (e.g. incorporating workload modeling), and it is our hope that this work instigates other work in this emerging area of research.

## REFERENCES

[1] SNIA Green Storage Initiative. *http://www.snia.org/forums/green*.

[2] VMware Infrastructure Architecture Overview White Paper. *http://www.vmware.com/pdf/vi_architecture_wp.pdf*.

[3] Report To Congress on Server and Data Center Energy Efficiency. In *U.S. EPA Technical Report*, 2007.

[4] Seven Strategies to Improve Datacenter Cooling Efficiency. White Paper #11, Version. 1.0, Green Grid, 2008.

[5] Power and Temperature Measurement Setup Guide SPECpower V1.1. *SPEC Power*, 2010.

[6] R. H. Arpaci-Dusseau, E. Anderson, N. Treuhaft, D. E. Culler, H. M. Hellerstein, D. A. Patterson, and K. Yelick. Cluster I/O with River: Making the Fast Case Common. In *IOPADS*, 1999.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP*, 2003.

[8] L. A. Barroso and U. Hölzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 40(12), 2007.

[9] L. A. Barroso and U. Holzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. *Synthesis Lectures on Computer Architecture*, 2009.

[10] C. Belady. In the Data Center, Power and Cooling Costs More than the IT Equipment it Supports. *Electronics Cooling*, 23(1), 2007.

[11] D. Bitton and J. Gray. Disk Shadowing. In *VLDB*, 1988.

[12] A. Borr. Transaction Monitoring in Encompass. In *VLDB*, 1981.

[13] K. G. Brill. Data Center Energy Efficiency and Productivity. In *The Uptime Institute - White Paper*, 2007.

[14] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving Disk Energy in Network Servers. In *Supercomputing*, 2003.

[15] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *NSDI*, 2005.

[16] D. J. DeWitt. The Wisconsin Benchmark: Past, Present, and Future. In J. Gray, editor, *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*. Morgan Kaufmann, 1993.

[17] X. Fan, W.-D. Weber, and L. A. Barroso. Power Provisioning for a Warehouse-sized Computer. In *ISCA*, 2007.

[18] J. Hamilton. Where Does Power Go In DCs and How To Get It Back? *Foo Camp*, 2008.

[19] J. Hamilton. Cooperative Expendable Micro-slice Servers (CEMS): Low Cost, Low Power Servers for Internet-Scale Services. In *CIDR*, 2009.

[20] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy Efficiency: The New Holy Grail of Database Management Systems Research. In *CIDR*, 2009.

[21] H.-I. Hsiao and D. J. DeWitt. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In *ICDE*, 1990.

[22] W. Lang and J. M. Patel. Towards Eco-friendly Database Management Systems. In *CIDR*, 2009.

[23] J. Leverich and C. Kozyrakis. On the Energy (In)efficiency of Hadoop Clusters. In *HotPower*, 2009.

[24] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In *ISCA*, 2008.

[25] D. Nelson, M. Ryan, S. DeVito, K. V. Ramesh, P. Vlasaty, B. Rucker, and B. Nelson. The Role of Modularity in Datacenter Design. *http://www.sun.com/storagetek/docs/EED.pdf*.

[26] A. Noll, A. Gal, and M. Franz. CellVM: A Homogeneous Virtual Machine Runtime System for a Heterogeneous Single-Chip Multiprocessor. In *Workshop on Cell Systems and Applications*, 2008.

[27] C. D. Patel and A. J. Shah. Cost Model for Planning, Development and Operation of a Datacenter. *HP Technical Report, HPL-2005-107R1*, 2005.

[28] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *SIGMOD*, 1988.

[29] PG&E. High Performance Datacenters. *http://hightech.lbl.gov/documents/DATA_CENTERS/06_DataCenters-PGE.pdf*.

[30] E. Pinheiro and R. Bianchini. Energy Conservation Techniques for Disk Array-Based Servers. In *ICS*, 2004.

[31] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In *Workshop on Compilers and Operating Systems for Low Power*, 2001.

[32] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting Redundancy to Conserve Energy in Storage Systems. In *SIGMETRICS*, 2006.

[33] K. Rajamani and C. Lefurgy. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In *Proc. of the IEEE Intl. Symp. on Performance Analysis of Systems and Software*, 2003.

[34] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level Power Management for Dense Blade Servers. In *ISCA*, 2006.

[35] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: a balanced energy-efficiency benchmark. In *SIGMOD*, 2007.

[36] S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza. Models and Metrics to Enable Energy-Efficiency Optimizations. *Computer*, 2007.

[37] S Greenberg and E Mills and B Tschudi. Best Practices for Datacenters: Lessons Learned from Benchmarking 22 Datacenters. *http://eetd.lbl.gov/EA/mills/emills/PUBS/PDF/ACEEE-datacenters.pdf*, 2006.

[38] S. Sankar, S. Gurumurthi, and M. R. Stan. Intra-disk Parallelism: An Idea Whose Time Has Come. In *ISCA*, 2008.

[39] S. Siddha, V. Pallipadi, and A. V. D. Ven. Getting Maximum Mileage Out of Tickless. In *Linux Symposium*, 2007.

[40] Teradata. DBC/1012 Database Computer System Manual Release 2.0. *Technical Document C10-0001-02*, 1985.

[41] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering Energy Proportionality with Non Energy-Proportional Systems - Optimizing the Ensemble. In *HotPower*, 2008.

[42] C. A. Waldspurger. Memory Resource Management in VMware ESX Server. In *OSDI*, 2002.

[43] C. Weddle, M. Oldham, J. Qian, A. Wang, P. Reiher, and G. Kuenning. PARAID: A gear-shifting power-aware RAID. *Trans. Storage*, 2007.