On Energy Management, Load Balancing and Replication

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

Abstract

In this paper we investigate some opportunities and challenges that arise in energy-aware computing in a cluster of servers running data-intensive workloads. We leverage the insight that servers in a cluster are often 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, and the energy cost as a component of the TCO is quickly rising [4, 7]. In addition, servers often run at low utilization, typically in the 20-30% range [3]. This low utilization suggests that one way of saving energy is to selectively power down servers. However, arbitrarily powering down servers that are running data-intensive applications is problematic, as it 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 mirroring replication strategy. To make this example more concrete, suppose that there are four nodes using mirrored replication, where each data partition is stored in exactly two different storage units. 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% of the provisioned 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% of the provisioned 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.

Given the many decades of work on designing replication schemes, the immediate question is whether or not there is a replication scheme that fits our goals of producing balanced, and energy-efficient cluster management strategies. As we will demonstrate, the surprising answer is yes — one of the earliest proposed parallel database data replication schemes, the "Chained Declustering" technique [10], when coupled with careful choices of which nodes to power down, can be exploited to achieve the above goal. In this paper, we present and evaluate two Chained Declustering-based schemes that differ in they way they power down/up nodes in a cluster.

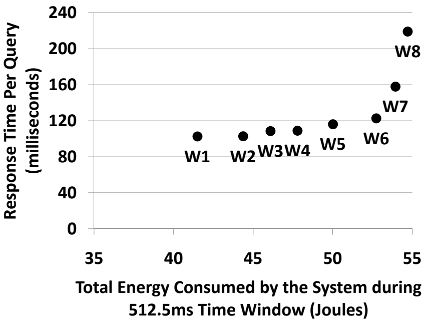


Figure 1. Energy consumption and response time profile

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.

2 Background and Problem Specification

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

2.1 Server Load vs. Energy Consumed

As pointed out in [3], the relationship between the load on a server and the energy consumed by the server is not linear. As an example, consider Figure 1, which shows the characteristics of a 1% clustered index query workload running on a commercial DBMS. (See Section 5.2 for more details about this workload.) 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 Wi 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 ($u < U$). Here U 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 $u' < U$ 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 U is defined relative to the initial designated peak load (cf. Section 2). Consequently, U can be greater than 100%; e.g., if the maximum tolerable response time is 120*ms* in Figure 1, then U is 120% (at W6).

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

In addition, 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.

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 1. An 8 node CD ring without failure.

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 (U) that they introduce. Some thoughts on picking appropriate values of U are presented in [11].

Finally, for certain system states, the nodes can be "perfectly balanced" – which means that each online node has the same node load. We discuss this further in Section 4.2.3.

3 Replication Revisited

Replication schemes are traditionally designed to allow 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, such as RAID [14], Mirrored Disk [5, 6], and Interleaved Declustering [20], 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. Our goal here 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.

Dealing with updates in this environment poses certain challenges, but our schemes can be adopted to handle updates, as discussed in [11].

3.1 Mirroring Replication

The basic principle used in mirroring [5, 6] is to make a second copy of the data and store it on a different storage device. Then, when some disk fails, the load on the remaining copies goes up dramatically. For example, consider 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, thereby doubling the load on the remaining disk. If a 2X increase in load is unacceptable, then with mirroring there is no energy savings if the system load is between 50 and 100%.

Also notice that with mirroring, there are only two operating states, 100% online nodes or 50% online nodes, which implies that it can't effectively adapt to loads in between these two extremes.

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	$\frac{8}{7}$	$\frac{8}{7}$	$\frac{8}{7}$	$\frac{8}{7}$	$\frac{8}{7}$	$\frac{8}{7}$	$\frac{8}{7}$

Table 2. An 8 node CD ring with 1 failure.

3.2 Chained Declustering (CD)

Chained Declustering [10] 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 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.

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 ($\frac{8}{7}^{th}$ of the original load) of queries, and hence is a *balanced* system.

While Table 2 shows what happens when one node is offline, CD can allow up to $N/2$ alternating nodes to go offline, where N is the number of nodes in the system. We exploit this property of CD to develop various energy management schemes.

4 Using Replication for Energy Management

While CD can tolerate a variety of configurations with nodes 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*.

A key observation is that if the **ring** or a **segment** of a CD set of nodes is broken, then the two new end nodes of the resulting segment(s) are **essential**, where the term essential for a node implies that removing that node makes the data unavailable. Thus, to take nodes offline any scheme must select additional nodes from the remaining online nodes that are not end points of the remaining segments. Next, we present two protocols for selecting which nodes to take offline.

4.1 Dissolving Chain (DC)

Using the key observation described above, the DC protocol sequentially withdraws nodes so that data is always available. DC starts with a full ring of nodes online, and when it takes the first node offline, it produced two segments of equal (or nearly equal) length. The next node it takes down is the middle node in the longest remaining segment.

At any given point in time, DC has a number of segments that it keeps sorted based on the segment length. Its powering down algorithm is then simple – simply take the middle node down in the current longest segment. More details about this method, including pseudocode and the node powering up sequence can be found in [11].

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".

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*

Notice that in DC once a node is powered down, that node continues to remain powered down if utilization decreases monotonically. This strategy can result in long segments, which in turn implies bigger variation in loads across the online nodes. The main intuition behind BC is to reduce these load variations by allowing powered down nodes to be brought online to make the current segments more uniform in terms of their lengths (which leads to lower load variations).

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, 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.*

Details about how to implement this optimal transition can be found in [11]. The following proposition 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 4.1. *$s(m)$ to $s(m')$ is an optimal Blinking Chain transition iff $(m|m'$ OR $m'|m)$*

See [11] for the proof.

Proposition 4.1 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 BC 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 BC 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 BC 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 $s(15)$. Details about implementing the Optimal BC Transition can be found in [11].

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*

Now consider the special states $s(m)$ where $m|N$, in which all the nodes have identical loads. We call such states **balanced** states, and denote this as $\bar{s}(m)$.

We can calculate the total number of possible balanced operating states for a Chained Declustered system of N nodes by recognizing that the number of primes in the factorization of N is what determines the number of balanced states. That is, we first factor N as $N = p_1^{N_1} p_2^{N_2} \ldots p_j^{N_j}$, where p_i is the i^{th} prime number. Then, 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.

5 Evaluation

To evaluate our proposed methods, 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.,

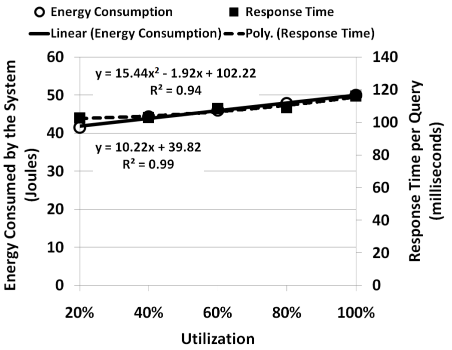


Figure 2. Index query regression model

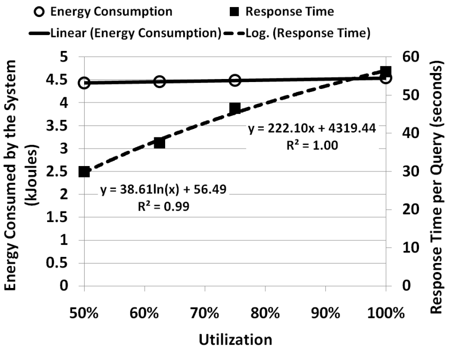


Figure 3. Database scan regression model

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.

In all results presented below, we consider a system with 1000 nodes ($N = 1000$).

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 WD Caviar SE16 320G SATA disk. The power supply unit was a Corsair VX450W. System energy draw was measured using a Yokogawa WT210 unit as suggested by [1].

We ran queries on a commercial DBMS against a Wisconsin Benchmark (WB) table [9]. Client applications accessing the database were written in Java 1.6 using the JDBC connection drivers for the commercial DBMS.

We ran each experiment five times, and report the average of the middle three results. The ACPI S4 state was used as the offline state. Other offline states are discussed in [11].

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 on a table with 20*M* 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.

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 20*M* 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 15*M* tuple table every 56.49*s*. 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 10*M* tuple WB table every 56.49*s*. Energy consumption is measured for the entire 56.49*s* window. With increased utilization, the increase in response time increases (nearly) linearly. All scans are “cold” and there is no caching between successive scans.

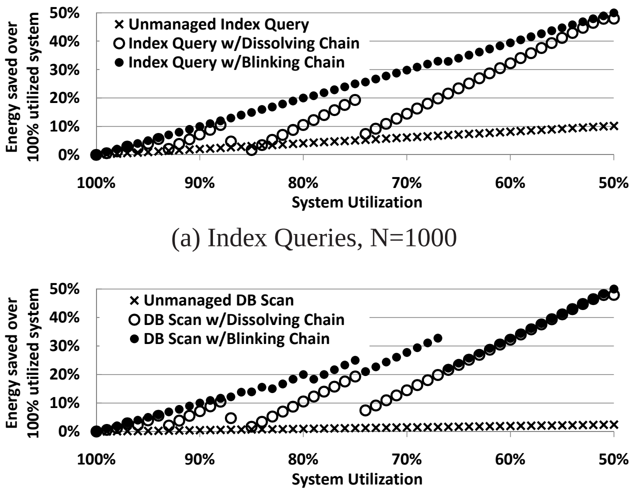
5.3 Modeling Energy and Response Time

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 average query response time as a function of utilization. The energy consumption model is linear while the response time model is quadratic.



(a) Index Queries, N=1000

(b) Scan Queries, N=1000

Figure 4. Energy savings v/s varying system utilization.

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.

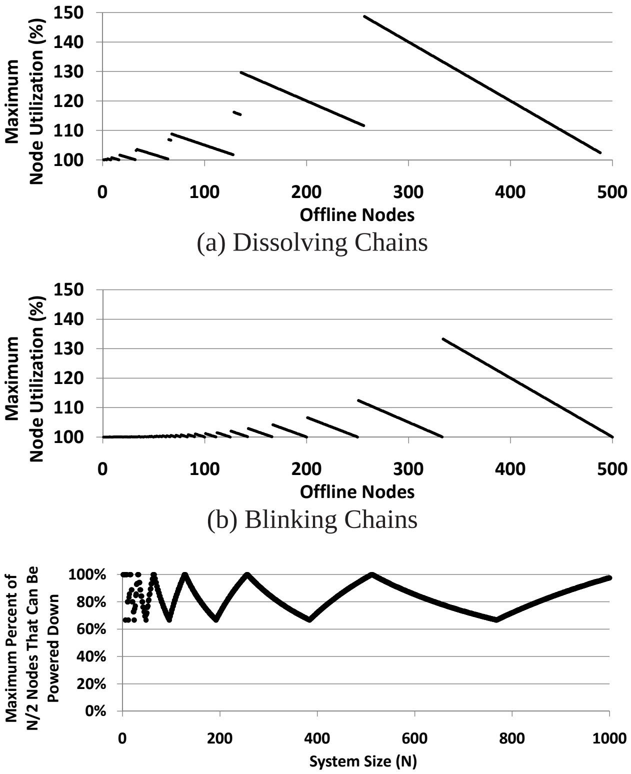
5.4 Effect of Decreasing Utilization

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.

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.



(a) Dissolving Chains

(b) Blinking Chains

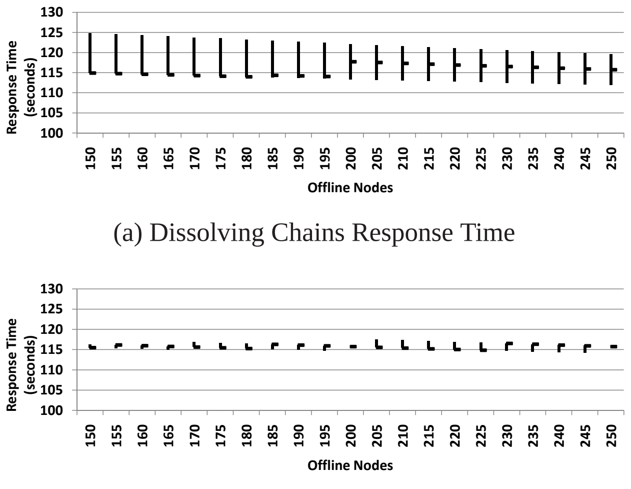
(c) Dissolving Chain maximum number off offline nodes

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

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



(a) Dissolving Chains Response Time

(b) Blinking Chains Response Time

Figure 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.

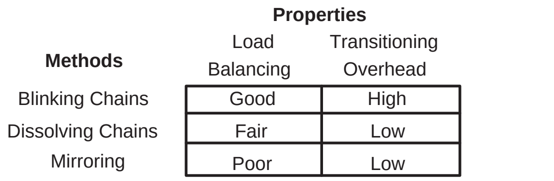
node utilization in imbalanced states (where maximum node utilization is greater than 100%) compared to DC.

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 $U = 120\%$, 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 U.

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 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.



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

Figure 7. Comparison of energy management methods

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 U in our problem statement (see Section 2.2).

5.5 Effect of Transitioning Costs

In the results above, we have not included any energy or latency costs associated with making transitions from one state to the next. 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. From the perspectives of energy consumed during the actual transitions, DC is clearly more efficient.

We have also examined the effect of the transitioning costs on BC. These results show that the BC transitions costs are acceptable if the system does not transition between states very often, and that for certain states DC will always be worse than BC. More details about this experiment can be found in [11]

5.6 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 7. 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

There has been considerable work on reducing the energy consumption of data centers, largely directed towards reducing the TCO [13], and include methods such as using more efficient power supply parts, raising data center temperatures, etc. These efforts are orthogonal to software methods for reducing energy consumption.

On the software systems side, a desired property is energy proportionality – i.e., an X% utilized server should consume X% of the peak 100% 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 [3]. For web servers, methods such as [16, 18] propose selectively turning servers off. Additional methods [15, 17] either rely on learning request skew, specialized hardware, and data migration and do not explore load imbalances caused by powering down disks.

Another mechanism for energy management is to use VM-based consolidation methods, such as [2, 8, 19, 21, 22]. However these methods are challenging to use for data-intensive applications as they may require moving large amounts of data during VM migration.

Recent work by Leverich powers down MapReduce cluster nodes but does not consider load balancing [12].

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 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%. One direction for future work is to build on the ideas proposed here and broaden the connections between generic levels of replication, load balancing 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.

8 References

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

[2] 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.

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

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

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

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

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

[8] 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.

[9] 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.

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

[11] W. Lang, J. M. Patel, and J. F. Naughton. On Energy Management, Load Balancing and Replication. Technical Report UW-CS-TR-1670, University of Wisconsin–Madison; Computer Sciences Department, 2010.

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

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

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

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

[16] 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.

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

[18] 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.

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

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

[21] 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.

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