Optimizing Parallel HPC Applications for Green Energy Sources

Cong Wang, Michael Zink and David Irwin
Department of Electrical and Computer Engineering
University of Massachusetts Amherst
{cwang, zink, irwin}@ecs.umass.edu

***Abstract*—**

While there has been significant prior research on optimizing the energy-efficiency of parallel applications, there has been much less on optimizing them for green energy sources, which expose rapid changes in power's availability (or cost) due to the use of local renewable energy (or utility demand response programs). In this paper, we present energy management policies that utilize active and inactive power capping to maximize the performance of rigid and elastic parallel tasks when subject to variable power constraints from green energy sources. We implement our policies on CloudLab, and evaluate their performance on multiple applications. Our results demonstrate the importance of designing for green energy with variable power. For example, we show that Graph500 requires 17% more time and 9% more energy to complete when power varies based on real-time electricity prices versus when power is unlimited at a fixed price. However, since real-time prices are lower than fixed prices, the total electricity cost of our best energy management policy when using real-time prices is 67% less than when using fixed prices.

I. Introduction

Energy consumption is widely regarded as the primary design constraint for scaling up the capacity of high performance computing (HPC) platforms, mainly due to the cost and carbon footprint of powering and cooling these systems [5]. To illustrate the magnitude of this constraint, recent estimates project that, if historical trends continue, the power requirements of an exascale platform in 2020 would be 200MW [5] with an annual electricity bill greater than $2.5B [4] primarily composed of "dirty" energy sources. These costs are unsustainable for even the highest-value applications. To reduce costs and carbon emissions, researchers continue to focus heavily on improving the energy-efficiency, i.e., the amount of computation done per joule, as evidenced by the "green" variants of the TOP500 [2] and Graph500 [1] rankings. Even so, reaching the current exascale design target of 20-40MW by 2020 will require an order of magnitude further improvement in energy-efficiency [13]. Achieving such improvements appears unrealistic in the near-term [23].

Thus, in this paper, we target a promising new direction for reducing energy costs and carbon emissions that focuses on optimizing the ability to rapidly and efficiently adapt parallel applications to dynamic changes in available power or cost. Our work envisions a tight coupling between the electric grid and large-scale computing platforms, enabling them to work in concert to efficiently balance electricity's supply and demand. Today's grid is grossly inefficient, primarily because, in most cases, utilities still balance supply and demand largely by only regulating supply, while granting consumers the freedom to *use as much power as they want, whenever they want.* This freedom imposes a steep price along multiple dimensions, resulting in wasted capital investments, high operational costs, excessive transmission losses, and limited renewable penetration.

The grid's inefficiencies have motivated recent "smart" grid initiatives to reduce peak demands and better handle intermittent renewables using demand-side management, which balances supply and demand, in part, by regulating electrical loads' energy usage, e.g., via real-time pricing or demand response. These initiatives offer consumers the potential for significantly lower costs, while also enabling consumers to use more green energy from renewable sources. As one example, Figure 1 plots electricity's real-time price (in New England's five-minute spot market) over a recent day to highlight that it varies dramatically even over short timescales. These fluctuations enable consumers to reduce costs by increasing energy usage when prices drop, and decreasing it when prices rise. In addition, the ability to adapt to power fluctuations also enables platforms to reduce their carbon emissions by increasing their reliance on local renewable energy sources, which generate energy intermittently based on changing environmental conditions. As outlined below, HPC platforms are well-suited to exploit such demand-side optimizations for four reasons.

- **Sophisticated Power Management**. HPC platforms already include advanced power management mechanisms that are remotely programmable, making them capable of rapidly and precisely varying their energy usage over a wide dynamic power range.
- **Delay-Tolerant Workloads**. Many HPC workloads are non-interactive batch jobs that are tolerant to delays in execution, providing them the flexibility to adjust their energy usage over time. The price elasticity of demand is higher for these workloads than many household and industrial loads, which are often interactive and not highly responsive to price fluctuations.
- **Large Energy Consumers**. The power requirements of future exascale-class HPC platforms (>20MW) will position them as some of largest industrial energy consumers with the highest electricity costs. As a result, they will have the most to gain from adapting their energy usage in response to changing grid conditions.
- **Rapid Growth Sector**. Despite energy-efficiency improvements, the power demands of HPC and data centers continue to rise, increasing by an estimated 56% from 2005-2010 and accounting for 1.7-2.2% of U.S. electricity usage [14], with usage expected to double every five years [25]. If current trends continue, these platforms

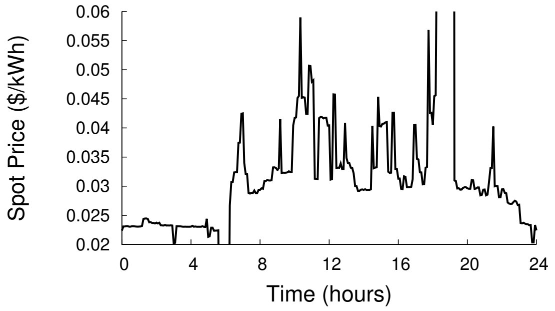


Fig. 1. Electricity's real-time price fluctuates significantly every few minutes.

will comprise 20% of U.S. electricity by 2030 [17]. Thus, regulating their demand holds significant potential to improve grid efficiency and stability, and reduce costs.

As we outline in Section VI, while researchers have recognized that, based on the properties above, HPC and data centers are ideal candidates for demand-side management, there has been little prior research on designing long-running, massively-parallel, and computationally-intensive HPC workloads for power variations. Thus, our goal is to provide insights into designing energy management policies that efficiently execute parallel tasks in the presence of time-varying, dynamic power constraints, which may derive from either the use of local renewable energy sources or participation in a utility demand response program. Our policies dynamically shift available power among a set of nodes executing a parallel application using active and inactive power capping techniques. Our hypothesis is that a policy that combines active power capping—to continuously reallocate available power among nodes based on their real-time utilization—with inactive power capping—to minimize the aggregate power consumption overhead—outperforms other policies in the design space. In evaluating our hypothesis, we make the following contributions.

- **Green Energy Challenges.** We outline the challenges associated with optimizing parallel applications for green energy sources with variable power. In particular, we focus on parallel applications that exhibit cross-node dependencies during execution, since these applications complicate both active and inactive power capping. We then present a model for a representative parallel application—based on a parallel breadth first search (P-BFS)—as a reference point for presenting our energy management policies.
- **Dynamic Energy Management Policies.** We propose dynamic energy management policies that utilize both active and inactive power capping to maximize parallel application performance subject to variable power constraints. We define dynamic and static policies that apply to both rigid parallel jobs, which can only utilize active power capping, and elastic parallel jobs, which can utilize both active and inactive power capping.
- **Implementation and Evaluation.** We implement our policies on a prototype cluster, and evaluate their performance using multiple applications. Our results demonstrate the importance of designing for variable power. For example, we show that the Graph500 benchmark requires 17% more time and 9% more energy to complete when power varies based on real-time electricity prices versus when unlimited power is available at a fixed price. However, since real-time prices are lower than fixed prices, the total cost of our best energy management policy when using real-time prices is 67% less than when using fixed prices.

II. Background

Our work assumes a parallel application that is subject to dynamic power constraints, which may arise from either the use of local renewable energy sources or participation in a utility demand response program. Our goal is to finish executing a parallel application as fast as possible given these dynamic constraints. More formally, we aim to minimize a parallel task's running time given a power signal $P(t)$, which dictates an average power cap over each interval $(t-\tau, t]$. Here, τ is the length of each interval, which we assume is dictated by the amount of energy storage capacity available. Since energy storage is expensive to install and maintain, smaller values of τ are better. The application defines its power constraint for each interval $(t, t+\tau]$ based on the energy it stored during the previous interval $(t-\tau, t]$. Thus, the power constraint at each interval is known at the beginning of the interval.

A. Power Capping Mechanisms

Given an average power budget each interval τ, we must distribute the available power among N nodes executing a parallel application. We leverage existing node power capping mechanisms, which ensure a node's power usage does not exceed a set threshold, to enforce a given distribution of power. Power capping may be either active or inactive. Active power capping uses Dynamic Voltage and Frequency Scaling (DVFS) or C-state throttling, which rapidly toggles processors between low-power idle C-states, to cap power without deactivating a node. The advantage of active power capping is that it keeps nodes active (albeit at a degraded performance level), is transparent to application software, is highly responsive (as power cap changes occur near instantly), and imposes a low overhead to transition power states. However, the primary disadvantage of active power capping is that it typically only targets CPU power, which accounts for only a fraction of node power usage, and thus its dynamic power range is limited. In practice, active power capping is typically only able to lower a node's power usage to at most 50% of its peak power [6].

In contrast, inactive power capping transitions nodes to an inactive state using either ACPI's Suspend-to-RAM (S3) or Suspend-to-Disk (S4) state. While inactive power capping is able to reduce a node's power usage to near zero, it incurs a high temporal overhead to transition states, e.g., tens of seconds for S3 and minutes for S4, and prevents any work allocated to an inactive node from making forward progress. The temporal overhead also wastes energy, since the node consumes energy while transitioning, but performs no computation. Importantly, inactive power capping is visible to applications, which must gracefully handle the dynamic loss and addition of nodes. Due to its high overhead, inactive power capping is only used when capping the power of clusters, since it enables a wider dynamic power range than when only using active power capping [24]. While inactive power capping is not as widely used as active power capping, we show that it improves performance under variable power relative to an approach that only uses active power capping.

B. Parallel Task Model

Our energy management policies described in Section III determine how to distribute the available power for each interval $[t - \tau, t)$ given the two power capping techniques above: active power capping, which may instantaneously adjust power consumption between some P_{max} and $P_{min} \geq\sim 0.50P_{max}$ while keeping a node active, and inactive power capping, which incurs some overhead T on the order of seconds to reduce power consumption to zero, but completely deactivates the node. The goal of our policies is to minimize the running time of a parallel task. Of course, parallel tasks may exhibit a wide variety of communication patterns and inter-node dependencies during their execution. Below, we illustrate key elements of our energy management policies using a simple representative example—a parallel breadth first search (P-BFS) algorithm—which serves as a building block for designing policies for tasks with more complex communication patterns.

P-BFS and other graph algorithms are a frequent subproblem in a variety of data-intensive HPC analytics applications, which is the primary reason it was chosen as the foundation of the recently introduced Graph500 benchmark [1]. In practice, P-BFSs are often massive in scope, searching graphs with tens of billions of edges and vertices on platforms with tens of thousands of cores [8]. Importantly, since the classic P-BFS [18] is "level-synchronous," it requires all-to-all communication among nodes at each level of the graph to determine whether a remote vertex has already been visited, i.e., by transmitting all remote edges on each node to the node that owns them, or transmitting all remote edges to a master node. Most non-embarrassingly parallel tasks will include similar types of barriers to synchronize their operation between phases.

While all-to-all synchronization barriers, which represent a dependency between each pair of nodes, are already the primary bottleneck for a P-BFS, consider the implications when using variable power. When using active power capping, reducing the power cap and performance level of one node will affect the performance of all other nodes, since to complete each level-synchronous step at each level of the graph, each node must communicate with all other nodes. However, setting all active power caps to the same value for each node may not guarantee the same per-node progress, since each nodes' progress depends on the portion of the input graph it is given at each level. As a result, each node's utilization and power will vary within each phase. Inactive power capping imposes a similar constraint, since a parallel task cannot continue until every node completes its work. Thus, overall progress is dictated by the slowest node to complete any phase.

Thus, the presence of both inter-node dependencies and unpredictable performance across nodes (due to differences in the partition of input data they receive) poses challenges when power varies. Any data-intensive application, such as P-BFS, which consists of a series of synchronization barriers will exhibit such inter-node dependencies.

III. Dynamic Energy Management Policies

In this section, we outline the design space for our energy management policies. We first propose policies that focus on rigid parallel applications, which only rely on active power capping. This is because rigid applications cannot adjust the number of nodes being used while the application is running. While active power capping affects performance, it is transparent to the application, which enables rigid applications to make use of it. Since inactive power capping deactivates nodes, it has the effect of periodically reducing the number of nodes an application is using; rigid applications cannot handle such reductions, as they will be perceived as failures. We then propose policies that leverage inactive power capping for elastic parallel applications. In addition to actively capping server power, these policies stretch and contract an elastic application while it is running by activating and deactivating nodes to maintain a platform-wide power cap.

A. Rigid Parallel Applications

Many parallel applications are rigid, including most MPI applications. While MPI 2.0 defines a dynamic process management scheme for elasticity, it is not commonly used. Assuming a parallel application, such as the P-BFS from the previous section, distributes the same amount of work to each node during each phase, and assuming each node is equivalent, the optimal policy to minimize running time is to equally distribute power among the nodes, such that each of the N nodes' active power cap is set to $P_{cap} = P_{available}(t)/N$ at each time t. We call this a *balanced* energy management policy. Since the completion time of each phase depends on all nodes in each phase completing, equally dividing the power among nodes, akin to load balancing, is the optimal policy. If any node were to receive less than this allocation, that node would serve as the bottleneck for completing each phase.

Unfortunately, in practice, nodes do not complete work at the same rate, largely due to differences in the partition of the input assigned to each node for each phase. For example, with a P-BFS, each node receives some partition of the graph, which it traverses. Since partitioning the work *a priori* into exactly equal partitions is not possible, node utilizations may fluctuate within a phase of execution based on the characteristics of their input data. For instance, one node may receive a dense subset of the graph, while another may receive a sparse subset. Other differences between nodes may also contribute to differences in per-node utilization, such as differences in computing capacity, e.g., 1.5Ghz versus 2.0Ghz, or in the set of background systems-level processes executing at any time. To illustrate, Figure 2 demonstrates the variance in power usage (due to differing node utilizations) for two equivalent nodes during the same execution of a large P-BFS using the reference MPI implementation of the Graph500 benchmark. Figure 2 demonstrates that node power usage, and thus node utilization, varies significantly between the nodes during execution, and does not proceed in lock-step.

As a result of this variance, we can improve upon the pure balanced policy above by employing a *dynamic* policy that continuously reallocates power within each phase based on node utilization. In contrast, the ideal balanced policy above is a *static* policy, since it adjusts node power caps based only on the available power and not node power usage. Our dynamic policy begins each interval by setting the power cap such that it equally divides power among nodes initially, but then measures node utilization and power at short intervals, e.g., every second, and adjusts active power caps based on node utilization levels. In particular, if any node is operating at less

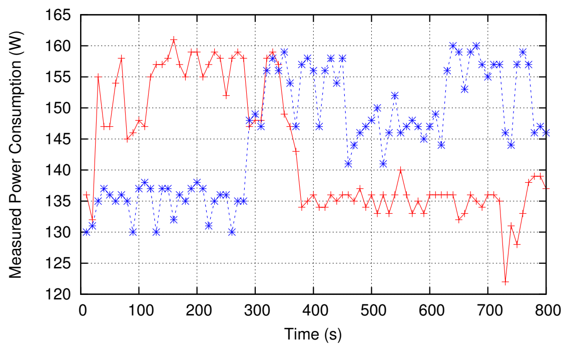


Fig. 2. Power fluctuations every 10 seconds for two worker nodes executing the reference MPI implementation of the Graph500 benchmark.

than K% CPU utilization, the policy reduces its active power cap such that it is above K% utilization, while progressively increasing the active power cap of nodes operating above K% utilization until they are below K% utilization. We set K near, but less than, 100% to account for our relatively coarse per-second measurement intervals; our prototype sets $K = 95\%$.

While our dynamic policy is aware of how CPU utilization changes with different active power caps based on profiling information known *a priori*, at high utilizations, it does not know how much more power a node near 100% utilization can effectively use. Thus, the policy progressively increases a node's power cap every few seconds by a configurable threshold, e.g., 10 watts (W). In addition, the policy equally distributes the power taken from nodes (and available for reallocation) among all nodes that are operating above 95% utilization. In essence, our dynamic variant of the balanced policy reduces wasted power by continuously (every second) taking power away from nodes that are not using it, and reallocating it to nodes that are using it.

B. Elastic Parallel Applications

As discussed above, elastic parallel applications can use inactive power capping, in addition to active power capping, to match available power. For example, if the available power is 50% of a platform's maximum power, then an elastic application may meet the power cap by deactivating 50% of the nodes, by setting the active power cap of each node to $P_{cap} = P_{max}/2$, or by some combination of the two techniques. Of course, inactive power capping incurs a high transition overhead—order of seconds-to-minutes depending on the platform—relative to active power capping, which is effectively instantaneous. Since the transition time represents overhead where a node consumes energy, but does no useful work, any use of inactive power capping must provide a benefit that exceeds the cost of the overhead.

A straightforward policy that uses inactive power capping is to greedily allocate available power at the beginning of each interval such that the number of active nodes $N_{active} = \lceil P_{available}/P_{max} \rceil$, where each node's active power cap is set to P_{max}. Such a policy could also be used in conjunction with the dynamic policy above to reallocate power within each interval by adjusting each node's active power cap. Relative to active power capping, deactivating nodes with inactive power capping is beneficial in reducing the fraction of the available power that contributes to an active node's overhead.

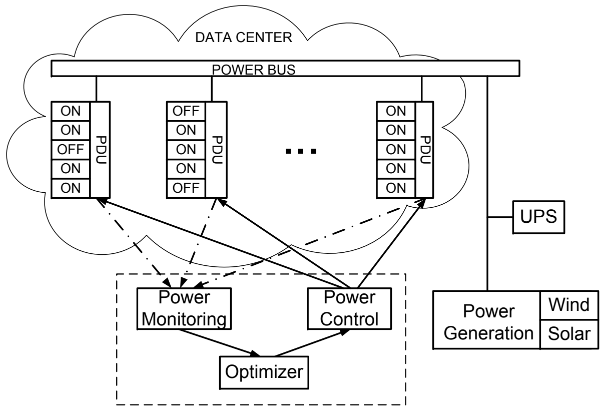


Fig. 3. Our prototype system architecture and power manager.

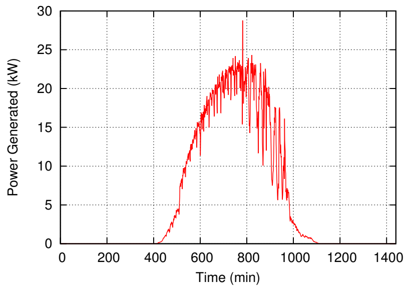
For example, since each node's idle power is roughly 50% peak power, simply activating a node without doing any useful work consumes a significant amount of power. This idle power consumption effectively represents wasted energy that is due to powering non-energy-proportional components, such as the motherboard, disk, NIC, memory, etc.

Thus, this idle power consumption represents *overhead power* for each active node, and the remaining power usage above idle represents *effective power*, where only the effective power contributes to actual program execution. While the dynamic balanced policy from the previous section works well if nodes cannot be deactivated, it incurs high overhead power across all nodes, since all nodes must remain active regardless of the available power. For example, consider a cluster with 10 nodes, each with an idle power of 100W and a maximum power of 200W. If available power is 1200W, the available power is 120W for each node, therefore, the total effective power for cluster is 200W, i.e., only 17% of the available power is being used effectively. By contrast, a *greedy* policy would only activate six nodes, which increases the fraction of effective power to 50%, and devotes more power to doing useful work. In this case, the greedy allocation, which uses inactive power capping, is significantly more energy-efficient and will result in higher performance and lower runtime.

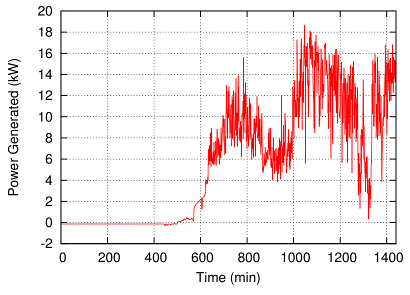
In addition to the greedy policy, to further reduce the overhead power, we define an *agile* policy that determines the optimal number of nodes to activate at the start of each interval. To do this, we compute the overhead power associated with each possible set of active nodes, from 1 to N, based on the available power. Here, we assume that each node operates at near 100% utilization; in this case, the policy activates the number of nodes that minimize overhead power. Once these nodes are active, the policy equally distributes the available power between the active nodes at the start of the interval; the policy may then employ the dynamic balanced policy among the active set of nodes within each interval. Of course, the length of each interval, the time to transition to and from the inactive state, and the frequency of these transitions dictates the overall transition overhead of our agile policy. We discuss the benefit of our agile policy for these parameters in Section V.

IV. Implementation

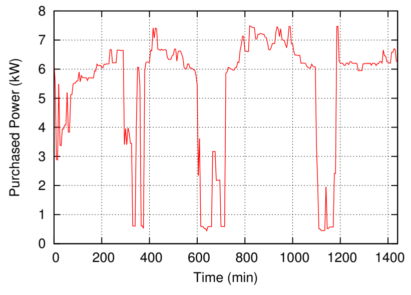
We develop a prototype power manager to implement the policies in the previous section. The power manager, depicted



(a)



(b)



(c)

Fig. 4. The average solar (4(a)) and wind power (4(b)) generated over a day, as well as a power signal based on using a fixed budget to purchase electricity at real-time prices in the five-minute spot market (4(c)).

in Figure 3, has three functions: monitoring power consumption and available power, executing our energy optimization policies, and deploying the policies by controlling power. The prototype monitors nodes using an out-of-band server management card that supports the IPMI protocol, which supports monitoring power at a 1Hz resolution at 1W granularity. We cap active power in our prototype by controlling CPU power states, e.g., via Dynamic Voltage and Frequency scaling and changing C-states, and capping utilization. Since the servers do not support inactive power capping via ACPI's Suspend-to-RAM (S3) state, we emulate S3 in our experiments by disconnecting nodes from the network.

We experiment with the performance of multiple parallel applications using our prototype and policies. Our primary application is the reference MPI implementation of the Graph500 benchmark [1], which implements a P-BFS and forms the basis for a large set of data-intensive parallel applications. By default, the MPI implementation of Graph500 is rigid, in that we cannot adjust the number of nodes it uses during its execution. However, we also experiment with an elastic variant of Graph500 by applying a method proposed by Raveendran et al. [19] to transform a rigid parallel task into an elastic one. In addition to Graph500, we also experiment with the Weather Research and Forecasting (WRF) application [3] and a Jacobi iteration application [11], which also use MPI. The WRF Model is a novel mesoscale numerical weather prediction application, while the Jacobi algorithm is a well-known numerical method for solving linear algebraic systems of n equations with n unknowns. The WRF power consumption behavior is relatively stable; while the communication pattern of Jacobi is similar to Graph500 with synchronization barriers that make power consumption more variable.

Our experiments utilize power signals from real solar and wind deployments, as well as signals based on real power prices from the wholesale electricity market. For each power signal, we select a representative day-long period with average power readings every minute.[1] The power signal for the solar and wind traces is shown in Figure 4(a) and Figure 4(b). For the energy price trace, we use the five-minute spot price from the New England Independent System Operator (ISO) on October 5, 2013 from 12am to 11:59pm. In this case, we assume our system has a fixed budget for purchasing energy, such that during a low price period it may purchase more power, and during a high price period it must purchase less.

[1]For solar power, we selected an early Fall day, September 28, 2014 from 12am to 11:59pm. For wind power, we selected a typical spring day, April 26, 2014 from 12am to 11:59pm.

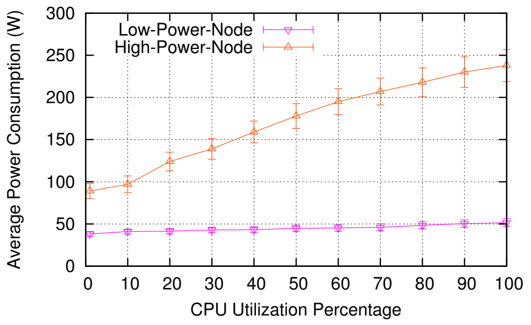


Fig. 5. Power usage at different CPU load levels in our prototype six node energy-agile cluster.

The resulting power signal is shown in Figure 4(c). Finally, to compare results with different power signals, we normalized all three traces such that they have the same average power.

V. Performance Evaluation

We evaluate our prototype in the CloudLab testbed [20] using two types of servers: the ARM-based HP Moonshot servers with 8 cores and 64GB memory, which are low-power nodes designed for energy-efficiency, and the Intel-based Dell R720 servers with 32 cores and 64GB memory, which are high-power nodes designed for maximum performance. The HP cluster consists of 65 nodes, while the Dell cluster consists of 6 nodes. Each of the two clusters are equipped with IPMI management card in order to monitor and control the real time power consumption of each node. Figure 5 shows the active power range of the two server types. As the figure shows, high-power nodes have a significantly larger active power range than low-power nodes. We evaluate both rigid and elastic scenarios. Recall that in our rigid scenario, our policies can only use active power capping, since they assume the application cannot activate and deactivate nodes, while our elastic policy is also able to use inactive power capping.

A. Rigid Parallel Applications

Figure 6 shows the runtime of Graph500 with solar, wind, and price-based power traces in our rigid scenario that only employs active power capping. The graph compares our static policy, which sets the power cap at the beginning of each interval based only on available power, with our dynamic balanced policy, which continuously shifts power based on which nodes are most effectively using it. As show in Figure 6, in our low-power cluster, our dynamic balanced policy reduces

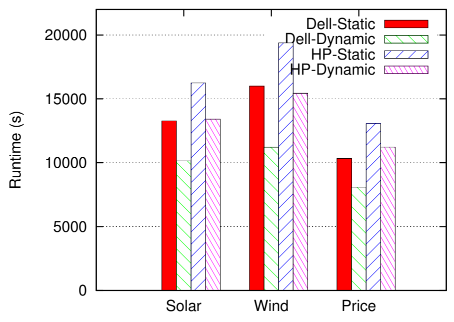


Fig. 6. Runtime of rigid Graph500 for solar, wind and price-based power signals using both the dynamic policy and static balanced policy that employ active power capping.

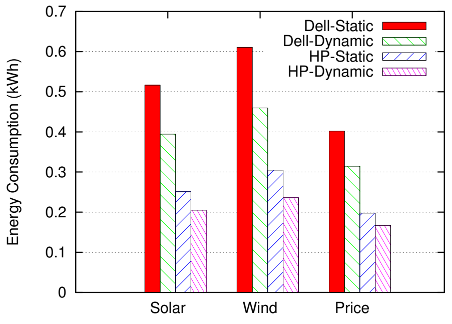


Fig. 7. Energy consumption of rigid Graph500 for solar, wind and price-based power signals using both the dynamic policy and static balanced policy that employs only active power capping.

the runtime of the Graph500 benchmark by 17% for our solar power trace, 20% for our wind power trace, and 13% for our price-based power trace, compared to static balanced policy. Likewise, in our high-power cluster, the dynamic balanced policy reduces the runtime by 23%, 29% and 21% for solar, wind and price-based power traces, respectively. Similarly, Figure 7 shows Graph500 consumes less overall energy when using a dynamic (as opposed to static) balanced policy.

The experiment demonstrates that adjusting power caps at fine-grained intervals can effectively increase the power allocation efficiency. By continuously reallocating power based on node utilization via adjusting the active power caps the application is able to make better use of the available power and improve its performance. The high-power cluster shows more improvement for two reasons: its servers i) exhibit a higher power variance between nodes, which provides more opportunity to adjust power, and ii) have a wider active power range, enabling them to reallocate more power between nodes. **Results:** *Our energy management policies decrease the energy consumption and improve the performance of a rigid Graph500 (by 13% to 29% for different nodes and power traces) by continuously directing power (using active power capping) to the nodes that can use it most effectively.*

B. Elastic Parallel Applications

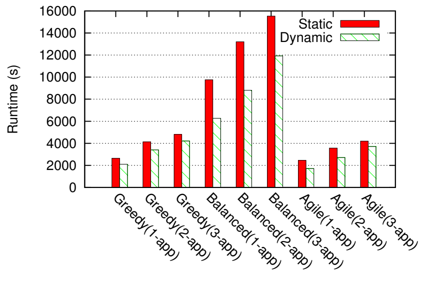
The results above are limited to rigid parallel applications that cannot handle activating and deactivating nodes while the application is running. Here, we demonstrate the performance of our energy management policies that employ active and inactive power capping. These experiments again use the solar, wind, and price-based power signals from Figure 4. However, we run multiple parallel applications, including Graph500, WRF, and the Jacobi solver. We show the performance of power policies when running a single application (Graph500), two applications (Graph500 + WRF), three applications (Graph500 + WRF + Jacobi), respectively. In all cases, we monitor the power, and apply the policies on the nodes regardless of the applications that are running. We examine the impact of elasticity, power variations, and energy storage on the performance in each case.

Impact of Elasticity. Figure 8 shows the runtime of three energy management policies: the dynamic balanced policy that is not elastic, and both our greedy and agile elastic policies, as described in Section III. Comparing the elastic policies in Figure 8 with the rigid (non-elastic) tasks in Figure 6 quantifies the benefit of elasticity: our elastic energy management policies achieve 41% less runtime for solar power signals, 36% less runtime for wind power signals, and 33% less runtime for price-based power signals. The benefit of elasticity derives from reducing the overhead power by deactivating nodes and concentrating more power on performing actual computation. While many parallel applications may not be elastic, our results indicate the importance of elasticity when designing for variable power. **Result:** *Energy management policies that are capable of elasticity, i.e., activating and deactivating entire nodes, further improve performance over rigid policies that are only capable of active power capping (by 33%-41% for our power signals).*

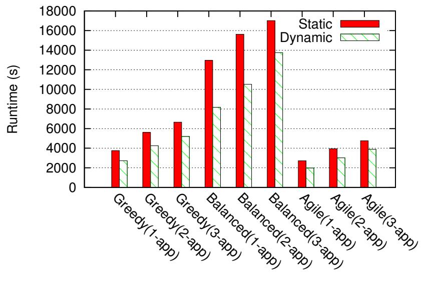
Impact of Power Variations. We next examine the impact of power variations on performance by comparing it in two scenarios: when power is stable and and when it is unlimited. For the stable power budget, we set a static power cap equal to the average power of the variable power signal. For the unlimited power budget, the nodes are free to use as much power as necessary. Figure 9(a) and Figure 9(b) show the runtime and energy consumption, respectively, of Graph500 when running *i)* with unlimited power and *ii)* with a fixed power cap equal to the average power.

As expected, when using variable power, as shown in Figure 8, the application takes longer to complete and consumes more energy than with unlimited power or with a stable power cap. In particular, in comparison to unlimited power, the greedy elastic policy takes up to 30% longer and uses 21% more energy, the balanced rigid policy takes up to 77% longer and uses up to 72% more energy, and the agile elastic policy takes up to 26% longer and uses up to 17% more energy. Likewise, comparing to a stable power cap, the greedy elastic policy takes up to 27% longer and uses up to 25% more energy, the balanced rigid policy takes up to 74% longer and uses up to 67% more energy, and the agile elastic policy takes up to 13% longer and uses up to 9% more energy.

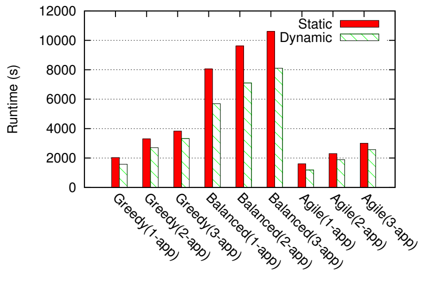
The results illustrate the importance of energy agile design for variable power. Since adapting to variable power introduces overheads that might cause an application to run longer than necessary, it typically uses more energy overall than when using unlimited or stable power. The goal of our policies is to limit this additional energy; results show that our agile elastic policy uses only 17% more energy than when using unlimited power and only 9% more energy than when using stable power. These comparisons are based on the worst case scenario, i.e., we compare the unlimited and stable power policies with the



(a)

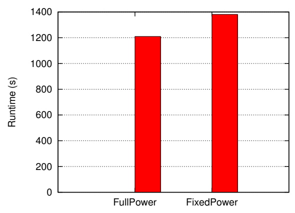


(b)

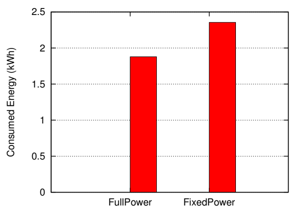


(c)

Fig. 8. Runtime of elastic variant of Graph500 for the greedy, balanced and agile policies for our solar (8(a)), wind (8(b)), and price-based power signals (8(c)).



(a)



(b)

Fig. 9. The runtime (9(a)) and power consumption (9(b)) of elastic Graph500 when operating under full power and fixed power budget.

Power Policy	Energy Cost (cents)
Full Power	21.27
Stable Power	27.44
Price Greedy	15.62
Price Balanced	42.07
Price Agile	12.77

TABLE I. COMPARISON OF ENERGY COST FOR NON-RENEWABLE POWERED CLUSTER.

longest runtime and the highest energy usage among the three power varying cases.

Based on the energy consumption data and energy prices in Figure 1, we are able to calculate the overall energy cost for our policies when using unlimited power and stable power, as well as using our price-based power signal with the greedy, balanced, and agile policies. The results in Table I show that although using unlimited power yields the shortest runtime and lowest energy consumption, it costs 36% more than using the greedy policy and 67% more than using the agile policy. This price advantage occurs because using unlimited power or stable power does not react to changes in price.
Result: *Adapting to a variable power source introduces overheads that increase application runtime and energy consumption (by 9%-17% in our experiments). Minimizing these overheads is important in quantifying the benefits of using a green energy source.*

Energy Storage. Our elastic policies are able to activate and deactivate subsets of nodes. However, as discussed earlier, activating and deactivating nodes imposes a long transition time along with the associated overhead power, which may impact overall performance. One way to reduce the number of transitions is to introduce energy storage capacity, which we capture using the interval τ over which power is known and stable. In Figure 10, we show how Graph500's runtime varies for different values of τ. Here, we assume a transition time of 10 seconds between active and inactive states, which is typical for transitions to and from ACPI's S3 state.

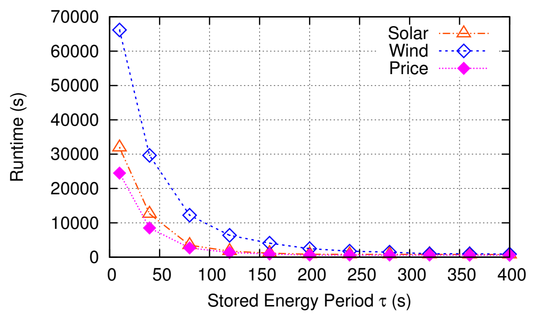


Fig. 10. Runtime of elastic Graph500 with different storage capacities (τ).

As shown, when τ increases, the application runtime decreases, due to fewer numbers of transitions. For example, in the wind power trace, the runtime decreases by $7\times$ when τ increases from 5 to 100 seconds. The results show that the energy storage capacity has a significant affect on performance, and that a relatively small amount of energy storage capacity (<150 seconds) can provide significant performance improvements. In particular, in this experiment, the runtime decreases to nearly the minimum for $\tau = 150$ seconds. Since storage is expensive to install and maintain, quantifying the effect of energy storage capacity on performance is important in assessing its costs and benefits. Our results show that a relatively small amount of energy storage capacity can reap most of the performance gains when using variable power.
Result: *A small amount of energy storage capacity (enough to support $\tau = 150$ seconds in our experiments) can significantly improves performance when using variable power sources.*

VI. Related Work

Recently, researchers have recognized the importance of optimizing the usage of variable power. This is due, in part, to a combination of rising energy prices and falling prices for solar panels and wind turbines. In particular, data centers are beginning to make use of substantial renewable deployments, as evidenced by the 40MW co-located solar farm that powers Apple's new iCloud data center in Maiden, North Carolina [7].

Initial research on optimization for power variations has focused on either using energy storage to offset power shortages [10], or enabling isolated system components to adapt their power usage. Research on using energy storage focuses on the best combination of energy storage devices, e.g., batteries, flywheels, etc., to minimize costs, and evaluates the potential savings based on realistic workloads, power prices, and battery costs. Our work differs from this work by focusing on optimizing parallel applications given a variable power source and a small fixed amount of energy storage capacity.

Another research direction is on adapting different system components to run on variable power. These system components include web servers [15], distributed caches [21], file systems [22], virtual machine migrations [16], and batch schedulers [9]. Our work focuses on parallel applications, rather than individual components. Prior work on batch schedulers is most closely related to our work. However, this work differs from ours in its focus on short batch tasks, which a scheduler may simply defer until enough power is available to run them. Another interesting approach is to run parallel tasks in virtual machines (VMs) and migrate them before deactivating nodes to consolidate workload. However, VMs introduce additional virtualization overheads that degrade performance, and the migration overhead is often large, especially for HPC applications that have large memory footprints. In addition, consolidating applications on a small subset of nodes may overload nodes and degrade performance, i.e., by causing memory thrashing.

The slack-based energy gear optimization leverages inter-node bottlenecks in MPI programs to improve energy-efficiency [12]. Our work differs from this approach by considering power variations from green energy sources, as well as both active and inactive power capping techniques.

VII. Conclusion

In this paper, we investigate energy management policies for parallel computing applications that run on green energy sources. We propose a variety of energy management policies that make use of both active and inactive power capping to maximize parallel application performance when running on power resources with variable constraints. Our policies include both static and dynamic power allocation variants that can be applied to both elastic and rigid (non-elastic) parallel jobs.

We implement our energy management policies on a cluster in the CloudLab testbed which consists of 65 nodes, and then evaluate their effectiveness using real solar, wind, and price-based power signals. Our results demonstrate the importance and effectiveness of designing for variable power. For example, we show that Graph500 requires 17% more time and 9% more energy to complete when power varies based on real-time electricity prices versus when power is unlimited at a fixed price. However, since real-time prices are lower than fixed prices, the total electricity cost of our energy-agile policy with real-time prices is 67% less than when using fixed prices.

VIII. Acknowledgement

This work was funded by the National Science Foundation under grants #1419199, #1331572, and #1422245.

References

[1] The Graph 500 List. http://www.graph500.org/.

[2] TOP500 Supercomputer Sites. http://www.top500.org/.

[3] The weather forecasting model. http://www.wrf-model.org.

[4] The Opportunities and Challenges of Exascale Computing: Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee. Technical report, U.S. Department of Energy, Office of Science, Fall 2010.

[5] U.S. Department of Energy, Office of Science. The Challenges of Exascale. http://science.energy.gov/ascr/research/scidac/exascale-challenges/, Accessed January 2015.

[6] D. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *SOSP*, October 2009.

[7] Apple and the Environment. http://www.apple.com/environment/renewable-energy/, Accessed January 2015.

[8] A. Buluç and K. Madduri. Parallel Breadth-First Search on Distributed Memory Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, November 2011.

[9] I. Goiri, W. Katsak, K. Le, T. Nguyen, and R. Bianchini. Parasol and GreenSwitch: Managing Datacenters Powered by Renewable Energy. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2013.

[10] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar. Benefits and Limitations of Tapping into Stored Energy for Datacenters. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*, June 2011.

[11] E. Jajaga and A. Kllobocishta. MPI parallel implementation of jacobi.

[12] N. Kappiah, V. W. Freeh, and D. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. In *Proceedings of Supercomputing,*, pages 33–33, Nov 2005.

[13] P. Kogge. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. Technical report, September 2008.

[14] J. Koomey. Data Center Electricity Use 2005 to 2010. Technical report, Analytics Press, August 2011.

[15] A. Krioukov, S. Alspaugh, P. Mohan, S. Dawson-Haggerty, D. E. Culler, and R. H. Katz. Design and Evaluation of an Energy Agile Computing Cluster. In *Technical Report UCB/EECS-2012-13, EECS Department, University of California, Berkeley*, January 2012.

[16] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of Supercomputing (SC)*, pages 22:1–22:12, 2011.

[17] A. Mansoor. Tapping the Full Potential of Efficiency Future Outlook - 2030. Technical report, Electric Power Research Institute (EPRI), August 2011.

[18] M. Quinn and N. Deo. Parallel Graph Algorithms. *ACM Computing Survey*, 16(3), 1984.

[19] A. Raveendran, T. Bicer, and G. Agrawal. A Framework for Elastic Execution of Existing MPI Programs. In *International Symposium on Parallel and Distributed Processing (IPDPS)*, May 2011.

[20] R. Ricci, E. Eide, and the CloudLab Team. Introducing CloudLab: scientific infrastructure for advancing cloud architectures and applications. 2014.

[21] N. Sharma, S. Barker, D. Irwin, and P. Shenoy. Blink: Managing Server Clusters on Intermittent Power. In *Proceedings of ASPLOS*, March 2011.

[22] N. Sharma, S. Barker, D. Irwin, and P. Shenoy. A Distributed File System for Intermittent Power. In *Proceedings of the International Green Computing Conference (IGCC)*, June 2013.

[23] H. Simon. Why we need Exascale and why we won't get there by 2020. In *Optical Interconnects Conference*, Santa Fe, New Mexico, May 2013.

[24] 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 *Proceedings of the Workshop on Power-aware Computing and Systems (HotPower)*, December 2008.

[25] U.S. Environmental Protection Agency, Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431. August 2007.