Dynamic Thermal Management for Distributed Systems

Andreas Weissel   Frank Bellosa
University of Erlangen, Department of Computer Science 4 (Operating Systems)
{weissel, bellosa}@cs.fau.de

Abstract

In modern data centers, the impact on the thermal properties by increased scale and power densities is enormous and poses new challenges on the designers of both computing as well as cooling systems. Control-theoretic techniques have proven to manage the heat dissipation and the temperature to avoid thermal emergencies, but are not aware of the task currently executing or its specific service requirements. In this work we investigate an approach to dynamic thermal management with respect to the demands of individual applications, users or services. We show that the energy consumption and the temperature can be determined on a fine grained level and without the need for measurement, using information from event monitors embedded in modern processors. We extend the well-known abstraction of resource containers to an infrastructure for transparent energy and temperature management in distributed systems. The processing of a request can be throttled to meet the thermal requirements of the system, even if machine boundaries are crossed, e.g. by remote procedure calls in a client/server relationship. With this facility, energy consumption can be accounted and the resulting heat generation be controlled precisely without the need for expensive hardware. Experiments on a Pentium 4 architecture show that energy and temperature are accurately determined and thermal limits for the individual CPU and the whole distributed system will not be exceeded. Use cases and important implications of our approach are discussed.

1. Introduction

The ever growing demand for higher performance is paralleled by a dramatic increase of the power consumption of cluster-based servers. At the same time, impressive advances have been made in the area of packaging and the level of integration of server units. The trend to increased processing power and ultra-dense packaging results in very high power densities that are a severe problem in today's data centers. Cooling technology is either designed for the maximum power consumption in a conservative approach or for typical sustained power. In the latter case, a trigger mechanism is provided to respond with a throttling of activity in order to guarantee a reliable operation of the device in case of a thermal emergency. This *dynamic thermal management* (DTM) reduces the cost for cooling by uniformly throttling the system. However, application-, user- or service-specific requirements are neglected by existing approaches to dynamic thermal management schemes.

This paper presents a new operating system abstraction to transparently account and control the energy usage of individual tasks in a distributed system. Event-monitoring counters embedded in modern processors are used to determine on-the-fly the power consumption and who has used the power in the system. With the specification of the cooling system (thermal resistance and capacitance), the temperature can be estimated without the need for measurement and used to trigger task-specific throttling. This is achieved by extending the well-known concept of resource containers [3] to a distributed system, similar to Cluster Reserves [2]. Our *energy containers* are globally identifiable and represent activities or tasks composed of several processes communicating over the network, e.g. in a client/server relationship. Forwarding the identifiers does not require extra messages as they are sent piggyback with the network traffic via IPv6 connections. The concept is transparent to the applications running on the modified operating system as well as to other (unmodified) operating systems.

Our approach allows individual temperature limits to be set for each server based on its location in the cluster and the position of the air-conditioning units. Energy estimation of incoming requests prior to execution is not necessary, because the estimation happens at execution time. This avoids a potential bottle-neck, since the estimation is done in parallel on all the nodes in the cluster. With cluster-wide temperature control the requirements on the cooling facilities can be relaxed, making over-provisioning unnecessary.

Power and temperature measurements of a small cluster of Pentium 4 systems running real-world applications demonstrate that event-driven dynamic thermal management can handle energy budgets of applications and services while keeping the temperature of each server below a critical limit by constraining the CPU activity.

Section 2 reviews related work. We detail our approach to dynamic thermal management and present the abstraction of energy containers in section 3. Further we describe our implementation in section 3.3 and present an evaluation in section 4. In section 5, we discuss use case scenarios and motivate future directions in section 6. Section 7 summarizes our results.

2. Related Work

Power and power density are becoming a major challenge in system design. Not only the power density on the chip level is rising exponentially [9], but also the problems for infrastructure level power supply and cooling [4]. The need for dynamic thermal management originates from the widening gap between maximum power and typical active power [17]. With this technique thermal design can assume moderate average power instead of maximum power thus reducing the cost for packaging and cooling.

Direct feedback-driven activity reduction uses temperature sensors or on-chip activity monitors to determine the thermal state of the chip and initiate a reduction of activity of individual units or the whole chip by reducing their execution rate. This approach was successfully applied to microprocessors by a feedback-driven reduction of the clock frequency or a throttling of the instruction cache [8, 13, 7, 15].

Another implementation of dynamic thermal management is *task level throttling*. CPU intensive tasks are said to be “hot” when they use more than a specific CPU activity over a period of time. When temperature reaches a critical level, hot tasks are candidates for throttling. In this way the system is idling and the CPU spends more time in the low-power state, so the temperature is decreased [12]. In contrast to direct feedback-driven activity reduction, task-specific throttling does not affect necessarily the performance of important activities like interrupt processing and the execution of tasks that do not contribute significantly to the power consumption of the system. We argue that CPU activity as a measure to identify “hot” processes is not accurate enough. Contemporary processors show a wide variation of the active power consumption, so a finer grained energy and temperature estimation is needed.

Architecture-level power simulators are useful to calculate thermal plots of the processor die like the Pentium 4 [9]. Thermal plots are essential for the placement of a temperature sensor supporting feedback-driven thermal management. Skadron et al. present *HotSpot*, an approach to modeling thermal behavior in architecture-level power/performance simulators [16]. HotSpot identifies the hottest micro-architectural units and allows to evaluate techniques for regulating on-chip temperature.

Performance monitoring counters have proved to offer valuable information in the field of performance analysis [1] and cache-affinity scheduling in multiprocessors [5, 19]. Now they become more and more attractive in the area of power management: the power/performance characteristics of a running thread can be determined on-line by reading of event counters. According to the thread’s patterns the scheduler can find the optimal thread-specific clock-speed in a time-sharing environment to save energy with just minor performance penalties [18]. We present an extension of these approaches to power characterization using performance monitoring counters in order to estimate and limit the temperature level.

Managing energy as a first class resource and sharing this resource among competing tasks according to user preferences was introduced in *ECOSystem* [21]. The *Currentcy* model [20] allows to allocate and account energy, and to enforce energy limits. ECOSystem/Currentcy is very similar to our resource container infrastructure and could easily be adapted to support dynamic thermal management.

Aron et al. present the concept of Cluster Reserves [2] to achieve performance isolation of certain tasks. A cluster reserve comprises resource containers on different machines which represent the same task, similar to the approach taken in this work. Cluster Reserves guarantee a certain minimal proportion of cluster resources to every class of requests or service class. This remains true even if the total system load induced by other requests is high. The distribution of resources to Cluster Reserves is computed by solving a constrained optimization problem.

Sharma et al. [14] show that it is necessary to monitor the temperature distribution within a server room (using a large number of temperature sensors) and to use workload placement policies in order to correct thermal imbalances. The temperature distribution and especially temperature peak values can be predicted by monitoring the server utilization and the current temperatures measured by the sensors.

3. Event-Driven Dynamic Thermal Management

Knowledge of the thermal status of the processor is an indispensable requirement for dynamic thermal management. Chip sets which allow the reading of a thermal diode embedded in modern processors cannot be used for fine-grained management because they don't allow a correlation between the originator of power consumption and the effect of heating. Furthermore, our experiments revealed that reading the thermal diodes of a typical Pentium4 board imposes significant overhead on the system. It takes 5.5 ms to retrieve the current temperature level via the system management bus (SMBus).

Our approach is based on the performance counters in today's processors to clearly identify "hot" processes, to estimate the processor's power consumption, and to amply determine the temperature of the chip given that the ambient temperature is either constant or can be measured occasionally. Contrary to the measurement approach described above, access to the performance counters is very fast, allowing a process-specific update on energy consumption during every timer interrupt.

For measurements we used a P4 2GHz motherboard (ASUS P4B266E, DDR RAM) instrumented with four–terminal precision resistors attached between the board and the 3.3V, 5V and 12V power supply. The voltage drop at the sense resistors was measured with an A/D-converter with up to 40000 samples per second and with a resolution of 256 steps.
The increasing complexity of modern processors (superscalar architecture, out of order execution, branch prediction, ...) demands a more elaborate procedure to estimate on-the-fly the power consumption. While it was sufficient for former architectures like Pentium II to calculate the percentage of CPU activity [12], we registered a wide variation of the active power consumption between 30 W and 51 W for the P4 architecture running a compute intensive task. Because there are high-power tasks that need about 70% more power than low-power tasks, CPU cycles are no longer a clear indicator for energy consumption.

Our approach to energy estimation is to correlate a processor-internal event to an amount of energy. As events being monitored correspond to specific activities, this correlation has linear characteristics. Therefore, we select several events which can be counted simultaneously and use a linear combination of these event counts to estimate the processor's energy consumption. The approach is presented in detail in [6].

The final set of events and corresponding coefficients did well for most of the test programs. The worst energy estimation is 30% too high while the estimation was less than 1% wrong for 11 of 25 test programs.

Currently, our implementation is constricted to thermal management of the processor, because the rest of the system architecture is not covered by any energy-specific monitoring counters. The presented thermal model is not intended to compete with dedicated power simulators. However it should provide sufficient accuracy to account on-the-fly CPU energy to an energy principal, to determine the thermal status of the processor and to support appropriate throttling mechanisms (e.g. by placing the CPU in HLT state until an interrupt occurs).

3.1 Energy Containers

Local Energy Containers. To manage energy as a first class resource we apply the abstraction of resource containers [3]. In contrast to accounting to processes or threads, this mechanism considers resource consumption on kernel-level as well as resources used by server processes working on behalf of clients. *Energy Containers* are a specialized form of resource containers that can account energy accurately and with respect to client-server relations. When a machine is running under energy pressure, processes are throttled according to the limits of the energy containers.

Energy containers form a hierarchical structure with the root container at the top. The energy consumption is accounted to the responsible container and to all of its parent containers up to the root. Hence, the root container indicates the total energy consumption of the system. If an accountable device is idle, its energy consumption is accounted to the container of the idle task.

The association between processes and containers can be established dynamically by special system calls to reflect changes in the workload of the processes. It is also possible to precisely account the energy consumption of a server to a client on a per-request basis. While a server is reading a new request from a file descriptor, an implicit update of its energy container binding is triggered. This approach automatically propagates resource bindings from client to server applications running on the same node if they communicate over pipes or sockets. Local energy containers are presented in detail in [6].

Global Energy Containers. To handle client/server relationships across machine boundaries we have extended the energy containers by global, unique identifiers that are valid in the whole distributed system. Each node maintains a mapping between ids and local energy containers. If processes on different nodes communicate over the network, the ids are sent piggyback with the network traffic. On the receiving side, the kernel extracts the id from each network packet and binds the receiving process to the id's (local) energy container. A global energy container is represented by the local containers on each node mapped to the same id (see figure 1). To retrieve the energy consumption or to set limits, each corresponding local container has to be accessed.

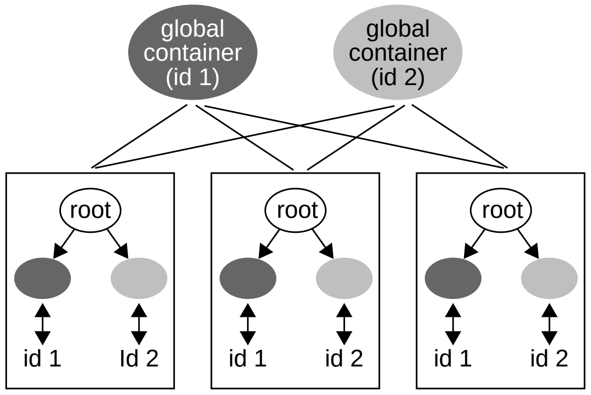

Figure 1. Global resource containers

A main design goal was transparency. The proposed infrastructure should not require modifications to user programs or network protocols. The modified kernel still needs to be able to communicate with hosts running unmodified kernels or other operating systems. The transparency of the system is vital for its application in real computing environments rather than restricting it to certain made up scenarios. Therefore we decided to base the communication on IPv6 and transmit the identifiers via optional IP header extensions.

The new protocol version IPv6 [10] has several advantages over IPv4, namely a much larger address space, simplified network administration by automatic host configuration, methods to provide secure communication (authentication headers, encapsulating security payload) and quality-of-service capabilities. However, the most important feature of IPv6 for our work are *destination options headers*. To avoid the overhead of very large headers due to 128 bit addresses and all the new capabilities, the header is divided into a mandatory and an optional part. One of the optional extensions are the so called destination options headers which are examined only by a packet's destination node. Up to now, no destination options are defined except for two padding options used for data alignment. For our approach we defined a new destination option for the exchange of information related to energy containers.

Outgoing network packets that are sent by a process with a resource binding to a global energy container are extended by a destination options header containing the container's id. On the receiver's side, incoming network packets containing the new destination option are bound to the same global energy container, i.e. to the local container mapped to this id.

On each node, an array maps ids to containers; a reference to the respective energy container is stored for each entry. Whenever the table is searched for an energy container that does not have an id, the container hierarchy is ascended until either a container is found without a parent (the root container) or until a container with an id is found. This way, child processes do not cause a problem for the identifier system. If necessary it is even possible to assign a single id only to the root container so that all processes on that machine will be sending out network packets containing that same id.

We have implemented our approach in Linux. In the Linux kernel, incoming as well as outgoing network packets are represented by the `sk_buff` data structure. This structure can contain a reference to an energy container. When a packet is sent by a process bound to a certain container, the `sk_buff`'s reference is set to the sending process' resource binding. On reception of a packet, the reference in the `sk_buff` can be read and used to set the receiving process' binding.

The `ip6_xmit` function was modified to add a destination options header to outgoing packets. Identifiers need to be added to every packet that is sent (and bound to a global container), as packets from different sources with possibly different identifiers can arrive in any order and change the binding of the receiving process. For incoming network packets, we provide a handler for the new option which determines the local energy container mapped to the id and stores a reference of this container in the `sk_buff` structure of the packet.

We added the new option `IPV6_SEND_OPTS` to the `setsockopt` system call to allow an application to control the sending of destination options headers in outgoing network packets via one of the application's sockets.

To manage global resource containers, we wrote a user space demon (`globalrcd`) running on all nodes in the distributed system to create energy containers with a specified id, to retrieve the energy consumption and to set energy and temperature limits.

Limiting Resource Usage. An energy container is used to control power consumption storing the used energy as well as a limit. We do not limit the amount of energy, but the rate of energy consumption. Thus, time is split up in epochs and a container has an energy limit per epoch. This limit is refreshed according to the current energy policy of the machine. Every activity that consumes energy reduces the available energy of some container. As energy containers are just a special kind of resource containers, they can account for multiple resources. A task is allowed to run if all resources are available (e.g., energy and network bandwidth). Our implementation currently accounts CPU time and energy consumption.

The operating system stops all activities that do not have enough energy in their energy container and enters low-power states to reduce power dissipation. By putting the CPU into a low-power state (e.g., HLT-state) for a short duration of time, it is possible to modulate the processor power consumption. Further potential throttling mechanisms are discussed in [7].

Due to the hierarchical structure of energy containers, there is a control loop of one container affecting all containers in the sub-tree. The top-level resource container controls any energy consuming activity in the complete system. By changing the amount of energy that is refreshed in this container, system-wide power consumption can be managed according to thermal requirements.

To protect server processes from clients with a very low energy quantum, they always retain their original resource container as backup. As long as the client's container provides enough energy, its budget is used and energy is accounted to the client. If the client runs out of energy, the server, working on behalf of the client, would have to wait until the client's container is refreshed. This would lead to a situation of priority inversion if other clients which have not exhausted their energy budget would have to wait for the server, too. Thus, we modified the resource binding mechanism: if the client's container runs out of resources, the server can be configured to fall back on his original container. Other solutions to the problem of priority inversion exist, but are outside the scope of this paper and will not be presented here.

3.2 From Energy to Temperature

With the processor's energy input known, we are able to estimate the processor temperature by looking at the thermal characteristics of the heat sink. The heat sink's energy input consists only of the energy consumed by the processor, and can be formulated as

$$cm\Delta T = \Delta Q = \int_{t_1}^{t_1+\Delta t} P(t)dt$$

c:	constant	ΔT:	heat sink's temperature increase
m:	mass of heat sink	ΔQ:	difference of inner energy
P:	CPU power usage		
Δt:	elapsed time		

which is transformed into

$$dT = \frac{1}{cm}Pdt = c_1 Pdt. \qquad (3.2.1)$$

The energy output of the heat sink is primarily due to convection and can be formulated as

$$\Delta Q = \frac{\alpha}{r} \cdot (T - T_0) \cdot t = cm\Delta T$$

r:	thermal resistance	T_0:	ambient temperature
α:	constant		

which is transformed into *Newton's Law of Cooling*:

$$dT = -c_2(T - T_0)dt. \qquad (3.2.2)$$

Energy output by heat radiation does not have to be considered because the temperature is quit low (< 60 ºC) and the aluminium surface has a low radiation emitting factor. In addition to that, leakage power influences the total power consumption. Though this effect is temperature dependent, it shows only little variation for the temperature ranges of our experiments (30–60 ºC) and is therefore treated as constant.

Together, these two formulas are used as an approach to estimate the processor temperature:

$$dT = [c_1 P - c_2(T - T_0)]dt. \qquad (3.2.3)$$

Solving this differential equation yields

$$T(t) = \frac{-\tilde{c}}{c_2} \cdot e^{-c_2 t} + \frac{c_1}{c_2} \cdot P + T_0. \qquad (3.2.4)$$

To find values for c_1, c_2 and T_0, we conducted two experiments:

To determine c_2, the raise in processor temperature on a sudden constant power consumption and a sudden reduction to HLT power was measured with the thermal diode on the processor die (see figure 2).

Our thermal model resulting in the differential equation simplifies the real thermal conditions as it assumes

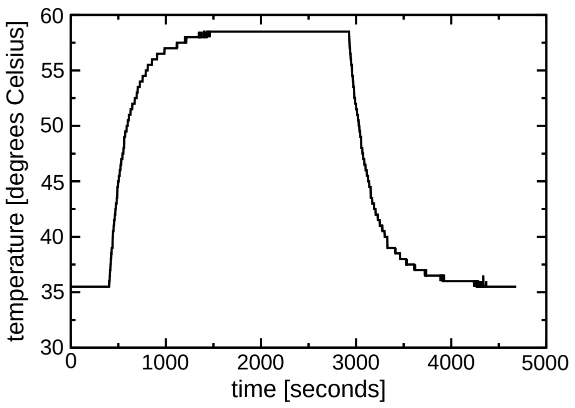

Figure 2. Temperature raise and decay due to constant power

a single heat sink interfaced with the chip without thermal resistance. However there is the impact of interface material like grease and phase-change films [17] and the thermal effects of heat spreaders. We found a simple approach to come close to the complex thermal model of several heat spreading components. We assume two constant values for c_2: $c_{2,up}$ for increasing temperature, and $c_{2,down}$ for decreasing temperature for modelling the overlay of the characteristics of interface material, heat spreader and heat sink in case of rising and falling temperatures: $c_{2,up} > c_{2,down}$ and $c_{2,up} \approx c_2 \approx c_{2,down}$.

With this approach, the estimated temperature is above the measured temperature in all of the test cases.

For c_1 and T_0, we measured the static temperatures and power consumption of the test programs and interpolated the resulting points with a quadratic function

$$T_s(P) = a_2P^2 + a_1P + a_0 \qquad (3.2.5)$$

which has to be above the curve measurements (see figure 3). Otherwise there could be a program for which the real static temperature is higher than the estimated. A quadratic function results in more accurate results than a linear fitting.

The values for c_1 and T_0 are computed from a tangent to $T_s(P)$ in the point $(P, T_s(P))$ with P being the current power consumption.

This results in the following differential equation to estimate the processor's change in temperature:

$$dT = [(a_2P + a_1)P + a_0 - T]c_2dt \qquad (3.2.6)$$

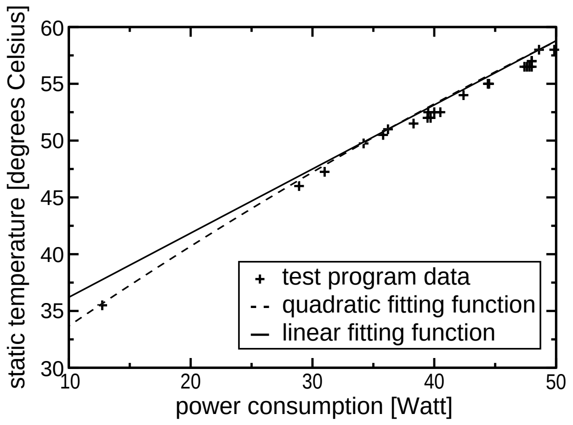

Figure 3. Estimating temperature for static power

with $c_2 = c_{2,up}$ if the last computed $dT \geq 0$,

else $c_2 = c_{2,down}$.

Once having measured a representative of the target systems, the four parameters just have to be loaded into the temperature estimation software. This estimator only needs information about the ambient temperature and a continuous flow of power values that can be determined with the help of the event-monitoring counters.

3.3 Implementation of Thermal Management

With Energy Containers, the kernel used in this work has the infrastructure for accounting and controlling energy consumption of processes and the entire machine. As a result, the temperature estimation and control could easily be implemented in user-space facilitating the use of floating point operations.

The resource container facility features simultaneous energy limits on different time-slices (128 ms and 1024 ms per default). Our approach to temperature control is to compute an energy limit for each time-slice for the whole computer (= root container), based on the current estimated temperature and the temperature limit. By limiting the root container's power consumption, the change in the processor's temperature (specified in equation (3.2.6)) will never result in an overrun of the critical temperature:

$$[(a_2P + a_1)P + a_0 - T]c_2dt \leq T_{limit} - T. \qquad (3.3.1)$$

Formula (3.3.1) forms the quadratic inequation

$$a_2P^2 + a_1P + a_0 - T \leq \frac{T_{limit} - T}{c_2dt}. \qquad (3.3.2)$$

Because $a_2 < 0$, the solution of this inequation is

$$P \leq \frac{-a_1}{2a_2} - \sqrt{\frac{1}{a_2}\left(\frac{T_{limit} - T}{c_2 dt} - a_0 + T + \frac{a_1^2}{4a_2}\right)}. \qquad (3.3.3)$$

We extended the utility *mbmon*, which reads the health monitoring chip set and displays the measured temperature, with the temperature estimator and the code to limit the root resource container. This eases calibration of the temperature estimation procedure. The energy consumption necessary for the temperature estimator is read from the root resource container. To prevent a deadlock, the mbmon-process is accounted, but never throttled.

Small errors in the temperature estimation mechanism or errors due to changing ambient temperature will accumulate over time. We measured an error of 3–5 ºC over a period of 24 hours. In order to prevent such deviations the estimated temperature is periodically adjusted to the measured temperature. For this re-calibration a period of 10 to 20 minutes is sufficient.

In order to examine the effects of energy- or temperature-aware process scheduling, we modified the allotment strategy for CPU time of the Linux scheduler. Originally, time slices are computed using the static priorities—the nice-levels—of the processes. We implemented a scheduler which computes time slices according to the relative power consumption of the process compared to the power consumption of the root container. This relation reflects the contribution of the process to the current power dissipation and, furthermore, to the current temperature level of the CPU. Additionally, the priority computation—the decision which process will run next—is based on the relative power consumption. With this approach "hot" processes are disadvantaged by the scheduler.

To sum up, we are able to identify hot processes using energy containers. We present two means to deal with them: first, limiting the power consumption of the attached containers automatically throttles hot processes as they spend their power budget faster than the others. Second, a power-based process scheduler can allot longer time slices to energy-efficient processes. While the second approach does not waste CPU time, throttling is needed to facilitate thermal management.

4. Evaluation

In [6], we evaluated the energy estimation and the temperature model of our approach by running several different applications and application benchmarks.

Over all test runs, the estimation error of the energy consumption was between 0% and 6%. The energy estimation of the root container is at least as high as the idle-power (about 13 W) that is accounted to the idle thread.

For all real-world applications the temperature estimator is within the accuracy of the temperature measurement (< 1 ºC). The worst-case scenario is an energy estimation error of 30% resulting in an error in temperature estimation of 7 ºC. So the error in temperature estimation is always the consequence of an error in energy estimation. The thermal model presented in section 3.2 did not show any shortcomings.

4.1 Distributed Energy Accounting

Figure 4 shows the power consumption and temperature of a small cluster of 3 servers. On the first server the apache web server *httpd* is running. Client requests to the cluster include database queries which are served by the database *postgreSQL* (server #3) and requests to factorize integers which are handled by a math software package (server #2). Apache uses PHP to access the services on the two other nodes of the cluster. A dispatcher node receives requests from two different clients. To account the energy consumed by the cluster on behalf of each client, two global energy containers are created. The dispatcher attributes the network packets with the corresponding container identifiers depending on the client ip addresses and forwards the packets to the server running apache. The figures show that the energy consumption of the three services is accounted to the two containers. As both clients issue the same requests to the database, both containers show roughly the same power profile on the third server. The reason for the high power consumption of container #1 on the second server is that larger numbers are factorized for client #1 than for client #2.

4.2 Enforcing Temperature Limits

Figure 5 shows the temperature and power consumption of two servers of our cluster, running apache and the factorization program, this time with a cluster-wide temperature limit of 47 ºC (shown by the shaded area). Mbmon wakes up periodically and computes the current temperature. The defined temperature level is enforced by setting an appropriate energy limit on the root container. As long as the computed temperature is below the target temperature, no energy limit is set. As can be seen in figure 5 the temperature of server #2 is rising up to 47 ºC. At time stamp 155 the energy consumption of the root

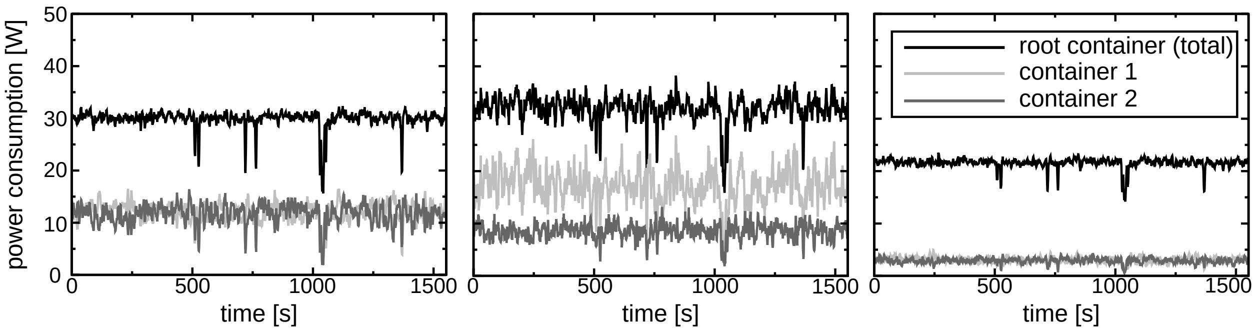

Figure 4. server #1 (apache) server #2 (factorization) server #3 (postgreSQL)

container of the second server is limited and the system is throttled in order to stay below the target temperature. The power profile of this server reflects the throttling of the root container. As the web server on node #1 has to wait longer for the results from the factorization, it is indirectly throttled, too.

Figure 6 shows another run of httpd. Initially no temperature limit is defined. Around time stamp 148s the limit is set to 50 ºC (again, the shaded area represents the allowed temperature level). This could be necessary if e.g. a cooling unit in a server cluster fails so that the cluster nodes have to be kept below a certain critical temperature. In our test, the thermal management needs about 110 seconds to reduce the temperature to the new limit.

The proof of concept is a web server accepting requests from two different classes of clients. When a critical temperature limit of 50 ºC is reached, client #1 should be preferred and should get a share of 80% of the allowed total power, while client #2 is just allowed to consume 20% of the remaining power. The power shares can be specified using user space tools which adjust the energy limits of the corresponding resource containers. Figure 7 shows the power consumption of the free running apache tasks working on behalf of the two classes of clients before and after reaching the predefined limit of 50 ºC. The root container reflects the sum of both client containers plus the power consumption of the halted CPU accounted to the idle thread.

4.3 Overhead

Reading of the event-monitoring counters is done in the timer interrupt (1000 times per second) or when a task is blocking. The context switching times in Linux 2.6 with energy container support is increased by 49% (5.9 µs) due to algorithmic overhead and the time for reading the event counters. However, for a typical scenario like kernel compiling we registered an overall performance loss of less than 1% (the time a kernel compile run needs on the original kernel compared to our modified kernel).

Estimating the temperature takes about 4.85 µs with a standard error of 0.843 because of a varying number of cache misses. Setting new limits to the root container requires 12.37 µs with a standard error of 1.537. The overhead for temperature estimation can be neglected because this procedure is typically executed 1–10 times per second. Furthermore, the overhead is by orders of magnitude smaller compared to reading the temperature sensors of the motherboard (which takes about 5.5 ms).

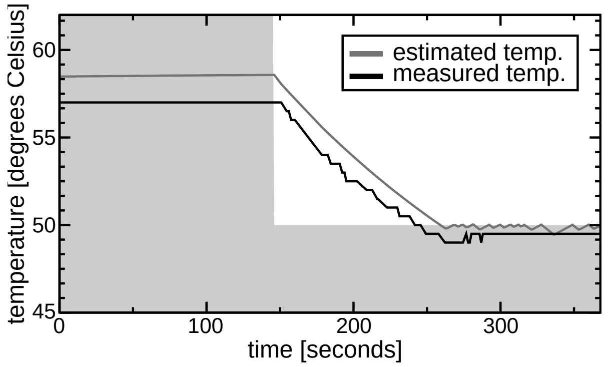

Figure 6. Cool-down with throttling

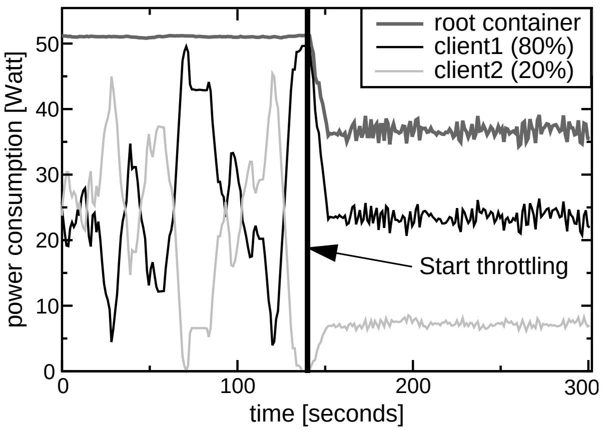

Figure 7. Throttling at 50 ºC according to energy shares

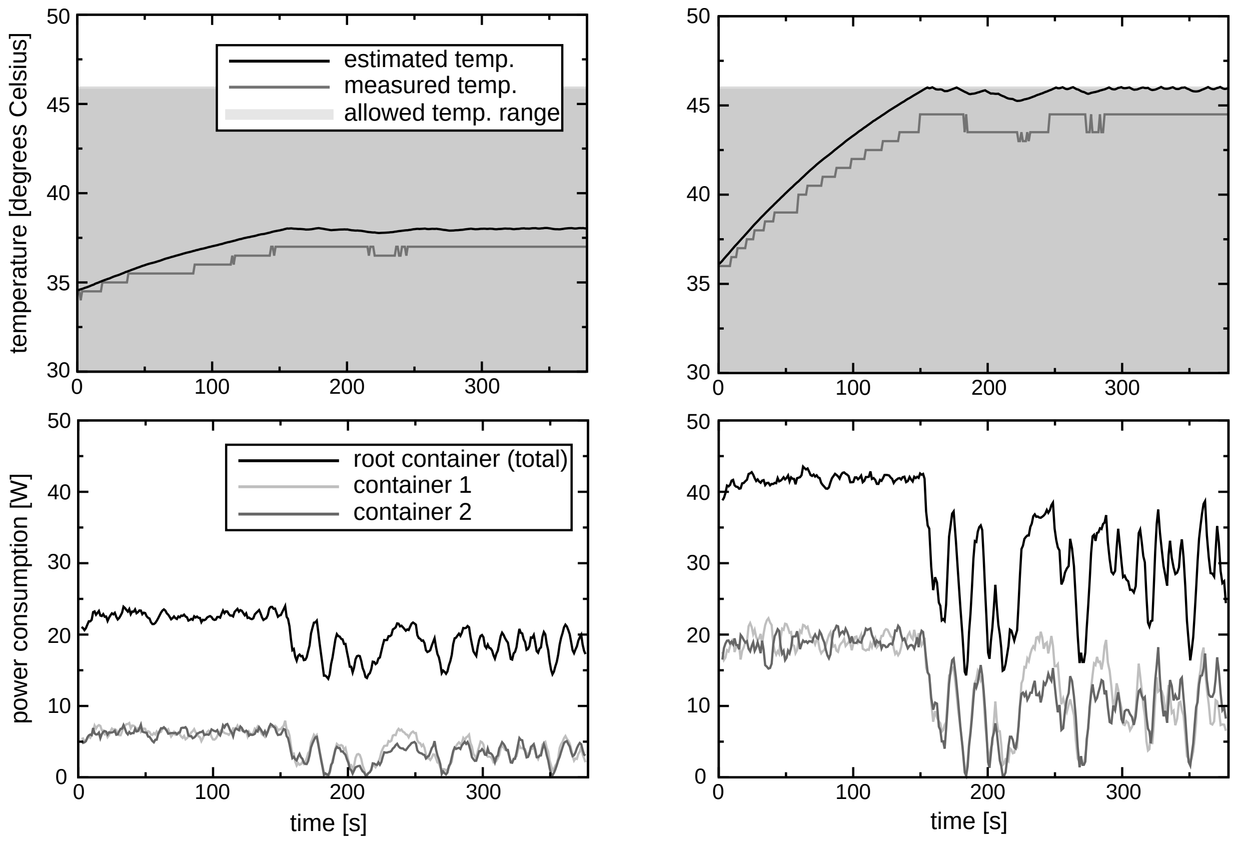

Figure 5. server #1 (apache), limit at 47 °C **server #2 (factorization), limit at 47 °C**

5. Use Case Scenarios

We have designed and implemented the infrastructure for task-level energy- and temperature management for distributed systems. In this section, we discuss implications and use case scenarios deploying this infrastructure.

If a global temperature limit is defined, there are several ways to translate it into local temperature and energy limits. Depending on the positions of the individual servers and the cooling units different strategies are possible. One would be to concentrate the load on a subset of servers so that the minimum number of cooling units has to be activated. When determining the set of servers and cooling units, the maximum temperature and the covered area of each cooling unit has to be taken into account. Another approach would be to uniformly distribute the temperature over the whole cluster. This way, a cluster-wide uniform temperature and power density can be achieved.

In the case of tasks with very short execution time, e.g. single web server requests, the dispatcher can forward incoming requests to the servers that have not yet exhausted their energy budgets. Thus, the dispatcher can react to unused energy on one server very fast by changing the workload distribution. However, long running tasks often cannot easily be migrated to servers with unused energy. The distribution of a task on several nodes in a cluster can lead to the situation that the global energy budget of this task is not fully used. In this case, our approach could be extended to collect unused energy from local containers and redistribute it among the servers that have exhausted their share already.

Implementation of such an extension is not straightforward, though. Our current approach simply discards unused resources after each epoch. Moreover, relocation of resources among the nodes would require extra network traffic as unused energy would have to be distributed quickly and frequently. Consequently, a significant overhead for such a relocation system is likely.

Up to now, we only transmit the identifiers of global energy containers across machine boundaries. It would also be possible to send information about the consumed energy piggyback with the response of the server to the client node. This way, an automatic accounting of the resource consumption of global containers could be implemented. Another possibility would be to transmit information about the temperature in the headers of the network packets, enabling a decentralized temperature management.

6. Future Directions

There is a multiplicity of interesting opportunities for research in operating systems and computer architecture. If the processor architecture allows a rapid change in clock frequency, task-specific frequency scaling is a further step to moderate the thermal load. Performance monitoring counters will provide the essential information for the power-performance trade-off. The thermal model has to be enhanced to deal with variable speed. Not only the number of events is relevant, but also the clock speed at which the events happen.

The architectural placement of counters and the types of countable events in today's computer architectures are devoted to performance profiling. In a hardware-/software co-design project we investigate the benefit of energy-monitoring counters (EMCs). In contrast to performance-monitoring counters, EMCs cover all energy relevant events. Furthermore the reading of these counters by the operating system is as fast as reading a processor register. The resulting low-overhead has to be paid by a loss in accuracy because we allow a delayed propagation of events to counters. By relaxing the timely resolution of the counters, we accelerate energy accounting and reduce the overhead in energy for managing the energy consumption.

Memory is becoming more and more a target for power management and energy accounting. According to precise energy estimation models for memory [11] we want to develop elaborate energy models for the use in operating systems that employ counters connected with the memory modules. These memory EMCs not only count read and write request but also the number of cycles the clock of the individual memory banks is enabled, and the number of cycles during which the rows in the individual memory banks are open.

7. Conclusions

We have presented an approach to dynamic thermal management with respect to the demands of individual applications, users or services. Based on the abstraction of resource containers, our approach provides the infrastructure for task-level energy accounting and temperature management for distributed systems. Energy accounting in a server cluster is implemented by attributing network communication, e.g. remote procedure calls in a client/server relationship, with information about the energy container. To meet the thermal requirements of the system, "hot" tasks (e.g. the processing of requests) are throttled by restricting their energy budget. Experiments on a small cluster show that energy and temperature are accurately determined and local and cluster-wide thermal limits are kept.

References

[1] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S.-T. Leung, R. Sites, M. Vandervoorde, C. Waldspurger, and W. Weihl. Continuous profiling: Where have all the cycles gone? *ACM Transactions on Computer Systems*, 15(4), November 1997.

[2] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. In *Measurement and Modeling of Computer Systems*, pages 90–101, 2000.

[3] Gaurav Banga, Peter Druschel, and Jeffrey Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating System Design and Implementation OSDI'99*, February 1999.

[4] Christian Belady. Cooling and power consideration for semiconductors into the next century. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01*, August 2001.

[5] Frank Bellosa and Martin Steckermeier. The performance implications of locality information usage in shared-memory multiprocessors. *Journal of Parallel and Distributed Computing*, 37(1):1–2, August 1996.

[6] Frank Bellosa, Simon Kellner, Martin Waitz, Andreas Weissel. Event-Driven Energy Accounting for Dynamic Thermal Management. In *Proceedings of the Fourth Workshop on Compilers and Operating Systems for Low Power COLP'03*, September 2003

[7] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings Of The Seventh International Symposium On High-Performance Computer Architecture (HPCA'01)*, January 2001.

[8] Christos J. Georgiou, Thor A. Larsen, and Eugen Schenfeld. Variable chip-clocking mechanism. United States Patent 5,189,314, February 1993.

[9] S. H. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, 2001. Q1 issue.

[10] Internet Protocol, Version 6 (IPv6), Specification. RFC 2460.

[11] Jeff Janzen. Calculating memory system power for DDR SDRAM. *Designline*, 10(2), 2001.

[12] E. Rohou and M. Smith. Dynamically managing processor temperature and power. In *Proceedings of the 2nd Workshop on Feedback-Directed Optimization*, November 1999.

[13] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, and J. Alvarez. Thermal management system for high performance PowerPC microprocessors. In *Proceedings of IEEE Compcon'97 Digest of Papers*, February 1997.

[14] Ratnesh K. Sharma, Cullen E. Bash, Chandrakant D. Pateland, Richard J. Friedrich, and Jeffrey S. Chase. Balance of power: dynamic thermal management for internet data centers. Technical Report HPL-2003-5, HP Labs, February 2003.

[15] Kevin Skadron, Tarek Abdelzaher, and Mircea R. Stan. Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. In *Proceedings Of The Seventh International Symposium On High-Performance Computer Architecture (HPCA'02)*, January 2002.

[16] Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th International Symposium on Computer Architecture (ISCA'03)*, June 2003.

[17] Ram Viswanath, Vijay Wakharkar, Abhay Watwe, and Vassou Lebonheur. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, 2000. Q3 issue.

[18] Andreas Weissel and Frank Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems CASES'02*, October 2002.

[19] Boris Weissman. Performance counters and state sharing annotations: a unified approach to thread locality. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS'98*, October 1998.

[20] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. Currentcy: Unifying policies for resource management. In *Proceedings of the USENIX 2003 Annual Technical Conference*, June 2003.

[21] Heng Zeng, Xiaobo Fan, Carla Ellis, Alvin Lebeck, and Amin Vahdat. Ecosystem: Managing energy as a first class operating system resource. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS'02*, October 2002.