2009 21st Euromicro Conference on Real-Time Systems

Hierarchical Utilization Control for Real-Time and Resilient Power Grid *

Ming Chen, Clinton Nolan, Xiaorui Wang, Sarina Adhikari, Fangxing Li, and Hairong Qi
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN 37996
{*mchen11, cnolan, xwang, sadhikar, fli6, hqi*}*@utk.edu*

Abstract

Blackouts in our daily life can be disastrous with enormous economic loss. Blackouts usually occur when appropriate corrective actions are not effectively taken for an initial contingency, resulting in a cascade failure. Therefore, it is critical to complete those tasks that are running power grid computing algorithms in the Energy Management System (EMS) in a timely manner to avoid blackouts. This problem can be formulated as guaranteeing end-to-end deadlines in a Distributed Real-time Embedded (DRE) system. However, existing work in power grid computing runs those tasks in an open-loop manner, which leads to poor guarantees on timeliness thus a high probability of blackouts. Furthermore, existing feedback scheduling algorithms in DRE systems cannot be directly adopted to handle with significantly different timescales of power grid computing tasks. In this paper, we propose a hierarchical control solution to guarantee the deadlines of those tasks in EMS by grouping them based on their characteristics. Our solution is based on well-established control theory for guaranteed control accuracy and system stability. Simulation results based on a realistic workload configuration demonstrate that our solution can guarantee timeliness for power grid computing and hence help to avoid blackouts.

1 Introduction

The power grid is a vital part of today's industrialized society, which relies upon the constant availability of high quality electrical power for proper functioning in industry, infrastructure, and domestic life. A power outage is likely to cause significant economic loss by disrupting the normal operating environment, which can be catastrophic. For example, the blackout that happened on August 14th, 2003 affected most of the northeastern United States and parts of Canada, and cost the United States between 4 billion USD and 10 billion USD [1].

Blackouts are typically caused by two or more contingencies that occur in a short time period. A contingency is an unplanned event that a power device is destroyed or degraded, *e.g.*, a transmission line is tripped or a generator is unexpectedly halted. A single contingency typically does not necessarily cause the system to become unstable because careful contingency screening and corrective actions (*i.e.*, a load shedding or a transmission loading relief) in EMS can dynamically re-adjust the system so that it is able to accommodate the single contingency. If the system stability can be ensured even after the worst single contingency occurs, the system is said to be in the *N-1 secure state* [2], where "N-1" means the normal system minus one key element. However, if a second contingency occurs before corrective actions are taken, the system may become unstable and result in a local blackout. Even worse, once the system has become unstable, more and more contingencies could occur as the system encounters conditions outside its operational range, which may result in a cascade blackout.

Clearly, it is critical to take corrective actions before a second contingency occurs to avoid blackouts. The North American Electric Reliability Council (NERC) has specified that once a contingency occurs, corrective actions must be taken within 30 minutes [3]. In a typical EMS, a group of algorithms should be completed in a timely manner to determine corrective actions: State Estimation, Contingency Screening, and Optimal Power Flow, which can be collectively referred to as *Real-Time Operation (RTO)* [4]. Its primary goal is to determine the proper corrective actions so that the power grid can stay in the N-1 secure state. First, *State Estimation (SE)* receives real-time data collected from sensors in the field by a Supervisory Control And Data Acquisition (SCADA) system. It then ensures that the data is correct and estimates any missing measurements, so that a complete and accurate model of the system is available. Second, if a contingency is occurring, *Contingency Screening (CS)* takes the estimated model and determines the constraints required so that no single contingency may lead to system instability. Finally, *Optimal Power Flow (OPF)* (also known as Economic Dispatch) assesses the security constraints and determines the most economically efficient way to set the states of all the devices. These settings are then sent to the SCADA, which forwards them to the field to fulfill the necessary corrective actions. Besides RTO, there are many other essential real-time tasks running in EMS.

*This work was supported, in part, by the US NSF under grants CNS-0831466, CNS-0716492, CNS-0720663, and CNS-0845390, and by the US ONR under grant N00014-09-1-0750.

978-0-7695-3724-5/09 $25.00 © 2009 IEEE
DOI 10.1109/ECRTS.2009.19

For example, *Automated Generation Control (AGC)* supervises and coordinates the power generators in real time [4]. *Short Term Operation (STO)* forecasts the customer power demand in the next hour and then optimally schedules the generation units. *Alarm Propagation (AP)* processes alarms to inform human operators of contingencies, so that corresponding measures can be taken manually to avoid system instability. Although these real-time tasks are not directly related to blackouts, continuously missing deadlines is detrimental to system stability. Most of the tasks introduced above are required to run on regular intervals in EMS by the NERC standard [3]. A list of typical real-time tasks, with their associated subtasks, in the EMS is presented in Table 1.

At present, many EMS run the power grid computing algorithms in an open-loop manner, so there is no guarantee that they will be completed by a given deadline. To guarantee the deadlines of those real-time tasks at runtime, several challenges must be faced. First, those real-time tasks are divided into multiple subtasks that are distributed among different processors in EMS and the execution of a task involves the execution of multiple subtasks under precedence constraints. To meet the end-to-end deadline, we need to meet the subdeadline of each subtask. Second, the computing system in EMS is an open and unpredictable environment. The workload in the system is system-dependent and time-varying, which cannot be accurately characterized a priori. Third, those tasks in the system have significantly different periods and execution times. For example, the period of AGC is usually in tens of seconds while the period of STO can be as long as thousands of seconds. Therefore, it is difficult to apply a single control algorithm to guarantee the deadline of all the tasks together.

To address those challenges in real-time power grid computing, we propose a novel control solution which features a hierarchical control architecture by differentiating the tasks with relatively long periods from the ones with relatively short periods. We explore the different properties of the collection of real-time tasks in power grid computing and apply different control algorithms to guarantee their deadlines. Specifically, the contributions of our work are four-fold:

- We address a real and important DRE application, *i.e.*, the power grid computing. We model power grid computing tasks as an end-to-end task model and apply real-time scheduling algorithms to guarantee the deadlines.
- We propose a novel solution which features a hierarchical control architecture to address the challenges of guaranteeing the deadlines of the power grid computing tasks and design different controllers to control tasks running on different timescales;
- We design and analyze the controllers based on well-established feedback control theory for theoretical guarantees on control accuracy and system stability.
- We present simulation results based on a realistic workload configuration to demonstrate that the deadlines can be guaranteed and the probability of blackouts is reduced.

The rest of the paper is organized as follows. Section 2 discusses the existing problems in power grid computing and proposes the control solution. Section 3 introduces the hierarchical control architecture. Section 4 presents the system modeling, design and analysis of the controllers. Section 5 introduces the simulation environment. Section 6 presents the results of our experiments. Section 7 discusses related work and Section 8 concludes the paper.

2 Existing Problems and Proposed Solution

In this section, we first introduce the problems in the current power grid by referring to the blackout on August 14th, 2003 as an example. We then give a high-level description of our proposed solution.

2.1 Problems of Current Power Grid

The Blackout on August 14th, 2003 is one of the most widespread electrical blackout in history. It affected approximately 40 million people in eight U.S. states and 10 million people in the Canadian province of Ontario. The blackout, according to the final report conducted by a US-Canadian joint task force [1], initially began with a trip of a transmission line due to tree contact. Unfortunately, the SE subtask in EMS had failed to assess the contingency periodically due to the combination of calculation errors and human faults. Therefore, the following CS subtask could not provide proper contingency screening and hence the system failed in taking corrective actions after the contingency occurred. In addition, the alarm system in EMS failed sometime shortly after the first contingency occurred, which prevented the human operator from taking effective measures manually. Heavier electric loads due to the former contingency led several transmission lines to sag lower so that more lines tripped due to tree contact. As a result, the contingencies followed one by one, and eventually caused a cascade blackout over a large area.

After an elaborate investigation, the joint task force emphasized the importance of the mandatory enforcement of those reliability standards specified by NERC[1]. In order to meet the NERC standards and improve the reliability of power grid computing, those tasks must run on regular intervals and meet their deadlines.

2.2 Proposed Solution

In this subsection, we first present an end-to-end task model commonly used for DRE systems such as power grid computing. We then describe our proposed solution.

As introduced in Section 1, the power grid computing tasks can be divided into multiple subtasks and each of them can run on different processors. The execution of a task involves the execution of multiple subtasks under precedence constraints (*i.e.*, CS runs after SE while OPF runs after CS.).

Table 1. Power grid computing tasks

	tasks	subtasks	execution time	period	type
1	Real-Time Operation (T_1)	State Estimation (T_{11})	$180s$	$500s$	long
		Contingency Screening (T_{12})	$100s$		
		Optimal Power Flow (T_{13})	$80s$		
2	Short Term Operation (T_2)	Load Forecasting (T_{21})	$120s$	$2000s$	long
		Unit Commitment (T_{22})	$120s$		
3	Automated Generation Control (T_3)	AGC Signal Request (T_{31})	$2s$	$[10s, 30s]$	short
		Automated Generation Control (T_{32})	$8s$		
4	Alarm Propagation (T_4)	Information Request (T_{41})	$2s$	$[4s, 20s]$	short
		Alarm Propagation (T_{42})	$2.5s$		
5	Voltage Security Assessment (T_5)	Voltage Security Assessment (T_{51})	$3s$	$[8s, 30s]$	short
6	Topology Analysis (T_6)	Topology Analysis (T_{61})	$4s$	$[8s, 30s]$	short

Each subtask also runs periodically and shares the same period as the task that it belongs to (*i.e.*, CS and OPF should have the same period as SE). Based on these facts, we can model those tasks as follows. A system is comprised of m end-to-end periodic tasks, $\{T_i | 1 \leq i \leq m\}$, executed on n processors, $\{P_i | 1 \leq i \leq n\}$. Task T_i is composed of a chain of subtasks, $\{T_{ij} | 1 \leq j \leq m_i\}$, that may be allocated to multiple processors. A subtask, T_{ij} $(1 < j \leq m_i)$, cannot be released for execution until its predecessor, T_{ij-1}, is completed. We assume that a non-greedy synchronization protocol (*e.g.*, release guard [5]) is used to enforce the precedence constraints between subsequent subtasks. Each subtask, T_{ij}, has an estimated execution time c_{ij} at the time of design. However, the actual execution time of a task may be significantly different from its estimation and varies at runtime.

Based on the task model above, the problem of meeting the deadline can be transformed into the problem of meeting the subdeadline of each subtask. A well-known approach for meeting the subdeadlines on a processor is by enforcing the *schedulable utilization bound* [5]. The subdeadlines of all the subtasks on a processor are guaranteed to be met if the utilization of the processor remains below its schedulable utilization bound. Previous work [6] has proposed an End-to-end Utilization CONtrol (EUCON) algorithm to guarantee the deadline of end-to-end tasks in a DRE system. EUCON is a Multiple-Input-Multiple-Output (MIMO) controller designed based on the Model Predicative Control (MPC) theory. It guarantees the deadlines of all tasks by controlling the CPU utilization of each processor to be within a certain schedulability bound by dynamically adjusting the rates of all the end-to-end tasks in the system. EUCON can provide robust utilization guarantees simultaneously on multiple processors when task execution times deviate from estimation or vary significantly at runtime. However, it cannot be directly adopted in power grid computing since that algorithm is designed with the assumption that the periods of all the tasks have the same timescale.

As shown in Table 1, the periods of the tasks in power grid computing can have significantly different timescales, which range from tens of seconds to thousands of seconds. In this paper, we refer to the tasks with relatively long periods as *long tasks* and the ones with relatively short periods as *short tasks*. Their subtasks are called *long subtasks* and *short subtasks*, respectively. The long subtasks can run iterative or non-iterative algorithms. A long iterative subtask uses an algorithm that achieves the specified computation accuracy after a certain number of iterations. We now discuss the characteristics of the tasks in power grid computing as follows:

- For short tasks, we assume that their rate can be dynamically adjusted within a range $[R_{min,i}, R_{max,i}]$. A short task running at a higher rate contributes a higher value to the application at the cost of higher CPU utilization. This is a reasonable assumption in power grid computing. For example, the more frequently AGC runs, the finer control it provides for power generation. Therefore, the adaptation of the rate adjustment can be incorporated into the CPU utilization control framework. In this paper, we use EUCON [6] to control the CPU utilization of short subtasks to a certain budget on multiple processors simultaneously by adjusting the rates of each short task. Hereinafter, we refer to it as the ***short task utilization controller***.
- For long iterative tasks, the number of iterations in the algorithm can be adjusted. The higher the accuracy it needs, the more iterations it must run at the cost of higher CPU utilization. In power grid computing, computation accuracy can be normally compromised for task timeliness. This is a valid assumption in that a slightly less accurate result is better than a result that misses its deadline. Therefore, the adaptation of the number of iterations of long iterative subtasks can be incorporated into the CPU utilization control framework. In this paper, we present a Single-Input-Single-Output (SISO) controller to control the CPU utilization of each long subtask to be within a certain budget. Since we only explore the property of the long iterative task in this work, *the long iterative task/subtask is referred to as the long task/subtask for simplicity* if not otherwise stated. Hereinafter, we refer to the controller of a long subtask as the ***long task utilization***

controller.

The two utilization controllers (*i.e.*, the short task utilization controller and the long task utilization controller) control the CPU utilizations of each long subtask and all short subtasks to their respective set points while the total schedulability utilization bound is enforced. Clearly, the more budget allocated to the long subtask from the total schedulibility utilization bound, the better computation accuracy the long subtask can achieve. However, this will lead to lower rates for the short subtasks and hence the less value they contribute to the system. Therefore, it is important to determine how the total schedulability bound should be optimally partitioned between short subtasks and long subtasks so that the long subtasks can achieve the specified computation accuracy while the short subtasks can run as frequently as possible to contribute more value to the system. In order to achieve the desired budget allocation, we propose an upper-level controller that controls the computation accuracy of each long subtask to the specified level by allocating the CPU utilization budget between the short tasks and long tasks. The total allocable budget is calculated by subtracting the utilization of the long non-iterative tasks from the total utilization bound. The variation of CPU utilization of the long non-iterative tasks is regarded as system noise in this paper. We refer to the upper-level controller as the ***accuracy controller***. In this paper, the ***computation accuracy*** is defined as the absolute value of the common logarithm of the computation error after each iteration. Hereinafter, we refer to the two utilization controllers as the lower-level controllers. The detailed system architecture is introduced in Section 3.

3 Hierarchical Control Architecture

In this section, we introduce the hierarchical control design. As shown in Figure 1, the two lower-level control loops, *i.e.*, the short task utilization control loop and the long task utilization control loop, control the CPU utilizations of short subtasks and long subtasks to their respective utilization budgets. The upper-level control loop, *i.e.*, the accuracy control loop, controls the computation accuracy of the long subtask by dynamically adjusting the CPU utilization budgets of the two lower-level control loops so that the total CPU utilization is guaranteed to approach the schedulability bound.

Long Task Utilization Control Loop. As shown in Figure 1, the key components in the long task utilization control loop include a utilization monitor, a long task utilization controller and an iteration modulator. The control loop is invoked periodically with a period selected to include multiple instances of the long subtask under control. At the end of every control period, the monitor first measures the CPU utilization contributed by the long subtask and sends the value to the long task utilization controller. The CPU utilization taken by the long subtask is the *controlled variable* in the control loop. The controller then calculates the

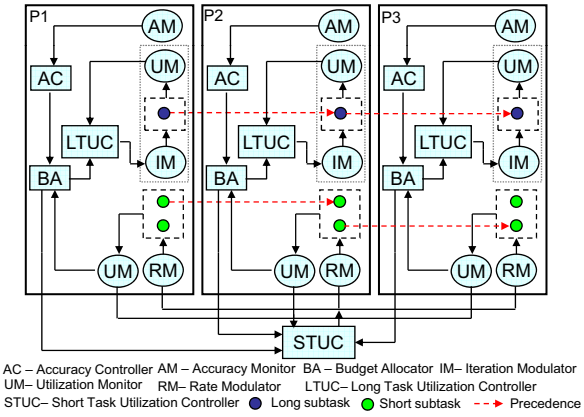


Figure 1. Hierarchical control architecture

maximum number of iterations that the long subtask can run for the next control period, based on the difference between the set point and the measured utilization. The number of iterations is the *manipulated variable*. Finally the iteration modulator notifies the long subtask of the maximum number of iterations when it creates new instances in the next control period. There is a long task utilization controller for each long subtask.

Short Task Utilization Control Loop. The short task utilization control loop maintains the desired CPU utilization of multiple processors simultaneously, despite the variation of execution time of the short subtasks at runtime. The key components in the short task utilization control loop include a centralized short task controller, a utilization monitor, and a rate modulator on each processor. The period of the control loop is selected so as to include multiple instances of the short task with the longest period. At the end of each control period, the utilization monitor on each processor measures the CPU utilization and sends the value to the centralized short task utilization controller. The CPU utilization of the short subtasks on each processor is the *controlled variable* in this control loop. The controller calculates the optimal change in the rate of each short task for the next control period and sends the value to each processor. The rate of each short task is the *manipulated variable* in this control loop. The rate modulator on each processor modifies the rates of the short tasks running on the processor, correspondingly. In this paper, we use EUCON [6] as the short task utilization controller. There is one centralized short task utilization controller in the whole system.

Accuracy Control Loop. The key components in the accuracy control loop include an accuracy monitor, an accuracy controller, and a budget arbitrator. The control loop is also invoked periodically. The period is selected to be longer than the maximum settling time of the two lower-level controllers. This guarantees that the two lower-level controllers can always settle to their respective utilization budget before the new budget is set, so that the control loops are decoupled and can be designed independently. At the end of every control period:

1. The accuracy monitor measures the average computa-

tion accuracy (*i.e.*, the absolute value of the common logarithm of the error) of all instances of the long subtask and sends the value to the accuracy controller. The computation accuracy of the long subtask is the *controlled variable* in the control loop.

2. The controller calculates the appropriate CPU budget for the long task utilization controller based on the difference between the measured accuracy and the set point (*i.e.*, the specified accuracy). The budget for the short task controller is calculated by subtracting the budget of the long task utilization controller and the measured utilization of the long non-iterative subtasks from the total schedulability bound. The two budgets are sent to the budget arbitrator. At the same time, the average utilization of the short subtasks in the last control period is also sent to the budget arbitrator. The budget of the long task utilization controller is the *manipulated variable* in this control loop.

3. The budget arbitrator compares the average utilization of the short subtasks with its budget to decide whether the short task utilization controller is saturated, which means the utilization of the short subtasks cannot settle to the set point due to the range of their rates. If the short task controller is saturated, the budget of the short task controller is set as the current average utilization and the budget of the long task controller is calculated by subtracting the budget of the short task controller and the measured utilization of long non-iterative subtasks from the total schedulability bound. If it is not saturated, the arbitrator does not modify the two budgets. The two budgets are then sent to the two lower-level controllers. The goal of the budget arbitrator is to prevent the accuracy controller from allocating too much or too little budget to the long subtask so that the short task utilization controller saturates, which may result in system over-utilization (*i.e.*, violation of the schedulability utilization bound) or system under-utilization.

There is an accuracy controller for each long subtask. The hierarchical control architecture can be extended to handle multiple long subtasks on the same processor by sharing the same budget arbitrator. In the case of saturation of the short task utilization controller, the budget arbitrator allocates the remaining budget proportionally to the requested budget of each accuracy controller.

Since the core of each control loop is its controller, we introduce the design and analysis of the controllers in the next section.

4 Controllers Design

In this section, we present the system modeling, design and analysis of the long task utilization controller and the accuracy controller. The design and analysis of the short task utilization controller can be found in [6].

4.1 Long Task Utilization Controller

We first introduce some notations. T_l is the control period. U_l is the set point. $p(k)$ is the period of the tasks/-subtasks. For long tasks, $p(k)$ is fixed at runtime. $it(k)$ is the number of iterations that a long subtask runs. $u_l(k)$ is the CPU utilization of the long subtask, and $exec(k)$ is the average execution time of the subtask in the k^{th} control period.

The goal of the long task utilization controller is to control the CPU utilization of the long subtask to approach the set point. In the meantime, the long subtask should run as many iterations as possible to achieve better accuracy.

System Modeling. In order to have an effective controller design, it is important to model the dynamics of the controlled system, namely the relationship between the controlled variables (*i.e.*, $u_l(k)$) and the manipulated variables (*i.e.*, $it(k)$). If the execution time of the long subtask is known, the CPU utilization that it takes is $u_l(k) = \frac{exec(k)}{p(k)}$. Since $exec(k)$ is proportional to the number of iterations that the long subtask runs, *i.e.*, $it(k)$, we have:

$$u_l(k) = \frac{exec\_per\_it(k)}{p(k)} it(k), \tag{1}$$

where $exec\_per\_it(k)$ is the execution time per iteration. Since the algorithm for the long subtask is known beforehand and we do not change the period of the long subtask at runtime, we can make the assumption that $exec\_per\_it(k) = exec\_per\_it$ and $p(k) = p$, where $exec\_per\_it$ and p are constants. As a result, we get the system model of the long task utilization controller as follows:

$$u_l(k) = k_u it(k-1), \tag{2}$$

where $k_u = \frac{exec\_per\_it}{p}$.

Controller Design and Analysis. Proportional-Integral (PI) control [7] can provide robust control performance despite considerable modeling errors. Based on the system model (2), we design a PI controller as follows:

$$it(k) = it(k-1) + K_1 e(k) - K_1 K_2 e(k-1), \tag{3}$$

where $e(k) = U_l - u_l(k)$ is the control error. Using the Root-Locus method [7], we can choose our control parameters as $K_1 = 1/k_u$ and $K_2 = 0$ such that our closed-loop transfer function is:

$$G(z) = z^{-1}. \tag{4}$$

Next, we reevaluate the control performance when the system (2) changes due to the variation of $exec\_per\_it$. Without loss of generality, we model the overall variation as g_u and get a real model at runtime:

$$u_l(k) = k'_u it(k), \tag{5}$$

where $k'_u = g_u \cdot k_u$. We apply the PI controller (3) on the real system model (5) to get the closed-loop transfer func-

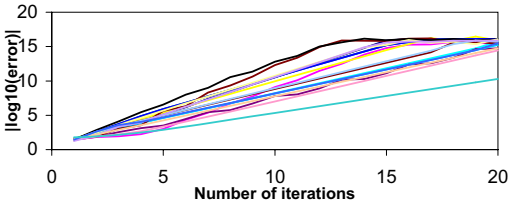


Figure 2. Linear relationship between the computation accuracy and the number of iterations

tion at runtime as:

$$G(z) = \frac{g_u}{z - (1 - g_u)}. \quad (6)$$

Based on control theory, we can prove that the closed-loop system at runtime (6) is stable and has zero steady-state error only if $0 < g_u < 2$. This analysis shows that the long task utilization controller (3) can effectively control the CPU utilization of the long subtask despite significant variations of $exec\_per\_it$, as long as those variations do not result in a k'_u that is twice the nominal value of k_u.

Our above analysis gives us theoretical confidence in the performance of our controller and provides a guideline to choose the parameters in the nominal system model (2). For example, given the possible minimum and maximum values of k'_u for a certain algorithm, we can choose the nominal k_u to be a value slightly larger than $\frac{k'_{u,min}+k'_{u,max}}{2}$, such that the system is guaranteed to be stable even when the real model is unknown at the design time.

4.2 Accuracy Controller

We first introduce some notations. T_a is the control period. R_a is the accuracy set point. $error(k)$ is the computation error of the long iterative task, and $a(k)$ is the computation accuracy of the long subtask. Specifically, $a(k) = |log10(error(k))|$. $b_s(k)$ and $b_l(k)$ are the budgets for the short task utilization controller and the long task utilization controller, respectively.

The goal of the accuracy controller is to control the computation accuracy of the long subtask to the set point, *i.e.*, the required accuracy, by dynamically adjusting the allocated CPU utilization budget to the long subtask on this processor. The more budget is allocated to the long subtask, the higher computation accuracy it achieves.

System Modeling. Similar to the long task utilization controller, we need to model the dynamics of the controlled system, namely the relationship between the controlled variables (*i.e.*, $a(k)$) and the manipulated variables (*i.e.*, $b_l(k)$). Unlike the long task utilization controller, the system model cannot be derived analytically. However, though the algorithms that the long subtasks run are highly system dependent (*e.g.*, [8] and [9] propose two different state estimation algorithms.), an important observation from our experiments is that there exhibits an approximately linear relationship between the computation accuracy and the number of iterations that the algorithm runs. For example, Figure 2 plots the relationship between the number of iterations and the computation accuracy based on different input information for one of the algorithms used in the subtask of SE. Although the curves in Figure 2 have different slopes, all are approximately linear. Based on this observation, it is valid to assume that there exists a model between the computation accuracy and the number of iterations as follows:

$$a(k) = k_a it(k-1), \quad (7)$$

where k_a is the nominal slope of the curve which models the relationship between the computation accuracy and the number of iterations. Since we already have the model between the CPU utilization and the number of iterations, presented in (2), we get a model between the utilization budget $b_l(k)$ and the computation accuracy $a(k)$ as follows:

$$a(k) = \frac{k_a}{k_u} b_l(k-1). \quad (8)$$

Controller Design and Analysis. Similar to the long task utilization controller, we design a PI controller as follows:

$$b_l(k) = b_l(k-1) + K_1 e(k) - K_1 K_2 e(k-1), \quad (9)$$

where $e(k) = R_a - a(k)$ is the control error, $K_1 = \frac{k_a}{k_u}$, and $K_2 = 0$.

If we model the variation of the parameter of k_a at runtime as g_a, we can depict the closed-loop system model at runtime as:

$$G(z) = \frac{g_a/g_u}{z - (1 - g_a/g_u)}. \quad (10)$$

Based on control theory, we can prove that the closed-loop system at runtime (10) is stable and has zero steady-state error only if $0 < g_a/g_u < 2$. Given an iterative algorithm, if we carefully choose the nominal values of k_a and k_u, we can make sure the closed-loop system is stable at runtime despite the variation of the computation accuracy after each iteration.

To handle models with k'_a and k'_u that are outside the established stability range, an online model estimator, implemented in our previous work [10], can be adopted to dynamically correct the models based on the relationship between controlled variables and manipulated variables such that system stability can be guaranteed despite significant variations.

5 Simulation Environment

Our simulation environment is developed based on the EUCON simulator [6] by adding the accuracy control loop and the long task utilization control loop. The simulator, the accuracy control loop, and the long task utilization control loop are implemented in C++, while the short task utilization control loop is implemented in MATLAB using the

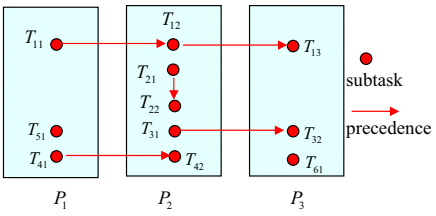


Figure 3. Workload configuration

lsqlin least squares solver.

Now we introduce the workload configuration, deadline assignment and period assignment as follows.

Workload Configuration. The configuration of those subtasks among the processors in EMS is highly system dependent. As an example, we configure the tasks listed in Table 1 among three processors in the simulator, which is shown in Figure 3. However, the hierarchical control solution can be easily reconfigured to handle different configurations. The subtasks on each processor are scheduled by the Rate Monotonic Scheduling (RMS) algorithm [5]. The precedence constraints among subtasks are enforced by the release guard protocol [5]. Since the algorithms of the subtasks of STO vary among different systems, we assume that they are long non-iterative subtasks for a typical setup in this paper. For example, [11] propose a non-iterative estimation scheme for the subtask of Load Forecasting. Since our objective is to highlight the hierarchical control solution, we also assume that there is only one long iterative subtask on each processor to simplify the setup. This means that there is only one long task utilization controller and accuracy controller on each processor. However, the hierarchical control solution can be easily extended to support multiple long subtasks on a processor as introduced in Section 3.

Deadline Assignment. The deadline of RTO should be within 30 minutes, according to the NERC standard [3]. However, the deadline assignment for other tasks is system dependent according to the power grid under control. In this work, we use a typical assignment, which is shown in Table 1. The end-to-end deadline of each task is evenly divided into subdeadlines for each subtask based on the number of subtasks that the task has.

In the simulator, when a subtask is ready to run, it first checks whether the task to which it belongs has missed the deadline or not. If the deadline has already been missed, the current instances of all subtasks in this task will be terminated and new instances will be started. This is important in power grid computing. Since the algorithm in the subtask may encounter dead loops due to unpredictable errors, continuing the task which has missed its deadline may lead to over-utilization of the processor and continuous deadline misses.

Period Assignment. In this work, we decide the period of each task based on their deadline requirement. According to the theory of RMS, each task's period $p_i = d_i/n_i$, where n_i is the number of subtasks in task T_i. d_i is the deadline of T_i. Therefore, the resultant period of each task T_i equals

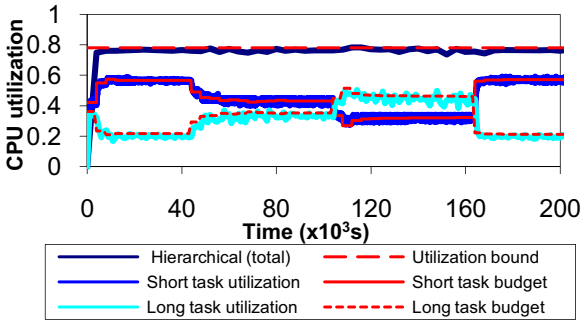


Figure 4. CPU utilization of the hierarchical control solution (P_1)

the subdeadline of its subtasks. Hence, the schedulable utilization bound of RMS is the total utilization bound on each processor [5]:

$$B_i = m_i(2^{1/m_i} - 1), \tag{11}$$

where m_i is the number of subtasks on P_i. All (sub)tasks meet their (sub)deadlines if the utilization bound on every processor is enforced. For example, P_1 in the setup shown in Figure 3 accommodates 3 subtasks and thus, its utilization bound is 0.7798.

Based on the task periods, we select the periods of the control loops as follows. The control period of the short task utilization control loop is set as $100s$ so that at least 4 instances of the short task are included. The control period of the long task utilization control loop and the accuracy control loop are set as $1,000s$ and $4,000s$ to include at least 2 and 8 instances of the task of RTO, respectively.

6 Experiments

In this section, we first demonstrate the performance of the hierarchical control solution, and then compare it with two baselines. In all the experiments, the simulation clock is equal to $10ms$ for higher precision. We only present the deadline miss ratio of the task of RTO. This is because its goal is to determine the corrective actions when a contingency occurs and continuous deadline misses of RTO may directly lead to blackouts.

6.1 Performance of the Hierarchical Control

In this experiment, one of the long subtasks increases the number of iterations required to achieve the specified accuracy at runtime. This is a common scenario to most power grid computing systems. For example, the computation accuracy of SE depends highly on the quality of the sensor data collected from the field. The greater the noise, the more number of iterations required to achieve the specified accuracy. Similarly, the computation accuracy of OPF relies on the complexity of the constraints estimated by CS. Without loss of generality, we choose SE to increase the required number of iterations at runtime in this experiment.

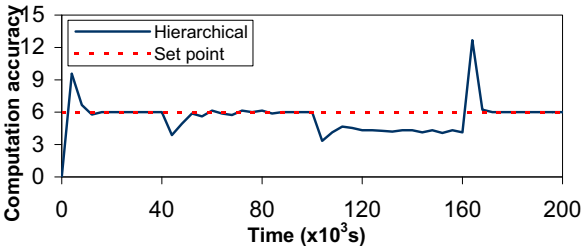


Figure 5. Computation accuracy of the hierarchical control solution (Subtask SE)

Figures 4 and 5 show the CPU utilization of the processor on which SE is running (*i.e.*, P_1) and the computation accuracy of SE, respectively. At the beginning, the required number of iterations is 6. We can see that the budget of the long task utilization controller decreases since the initial budget is allocated by assuming that the required number of iterations is 10. At time $40,000s$, the required number of iterations increases from 6 to 10, due to the degraded quality of the sensor data. The budget allocation between the long task utilization controller and the short task utilization controller is adjusted by the accuracy controller correspondingly so that the specified accuracy of 6 is still achieved (as shown in Figure 5). At time $100,000s$, the required number of iterations further increases to 20 due to the continuously degraded quality of the data. To achieve the specified accuracy, the accuracy controller further increases the budget of the long subtask and decreases the budget of the short subtasks, making the short utilization controller reach saturate. However, the budget arbitrator detects the saturation and throttles the budget of the short task utilization controller with the average utilization of the short subtasks in the last control period. Hence, the long subtask cannot get adequate budget and has to terminate the iteration before it achieves the specified accuracy. This results in a computation accuracy of 4.3 as shown in Figure 5. At time $160,000s$, the quality of data returns to the normal level and the required number of iterations decreases to 6. The budget allocation is adjusted and the long subtask achieves the specified accuracy again. During the whole run, the two lower-level utilization controllers effectively control the utilizations of the long subtask (*i.e.*, SE) and the short subtasks to approach their respective set points. As a result, the total schedulability utilization bound is guaranteed so that deadline misses are avoided.

Based on the results, we can conclude that the hierarchical control solution can dynamically adjust budget allocation between the long subtask and the short subtasks in response to variations, and effectively guarantee the total utilization bound to help avoid blackouts.

6.2 Comparison with OPEN

In this subsection, we compare the hierarchical control solution with the first baseline: OPEN, in term of deadline guarantees.

OPEN is an open-loop algorithm that employs fixed rates for the short tasks, and runs long subtasks until the specified computation accuracy is achieved. OPEN first estimates the number of iterations that the long subtask needs to run to achieve the specified accuracy. It then allocates the utilization budgets between the long subtask and the short subtasks in a static way. Based on the allocated budgets, it calculates the periods of the short tasks a priori based on the estimated execution time. OPEN can achieve the utilization bound and maximize the system performance (*e.g.*, running the short tasks as frequently as possible) if all of the estimated information is accurate.

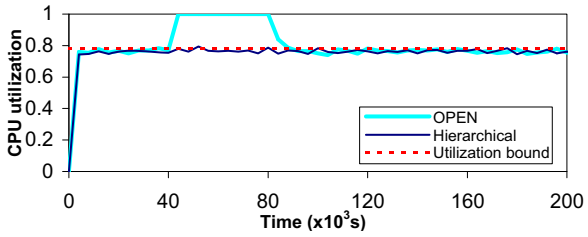
Similar to the scenario in Section 6.1, the required number of iterations to achieve the specified accuracy of 6 for the subtask of SE has a sharp increase from 10 to 20 at time $40,000s$. Figure 6(a) shows the total utilization of the processor on which SE is running (*i.e.*, P_1) for both OPEN and the hierarchical control. Figure 6(b) shows the miss ratio for the task of RTO which SE is a subtask of. As shown in Figure 6(a), OPEN indeed achieves the utilization bound at the beginning. However, OPEN keeps violating the total utilization bound after the abrupt variation occurs until the quality of the sensor data returns to the normal level at time $80,000s$. This is because OPEN has no feedback information from this abrupt variation and the long subtask of SE still runs until the specified accuracy is achieved. As a result, the task of RTO continuously misses its deadline due to the violation of the total utilization bound as shown in Figure 6(b). This is a very dangerous situation for the power grid. If a contingency occurs during this time, the power grid is vulnerable to more contingencies since corrective actions cannot be taken timely. As a result, there will be a high probability of a blackout (*e.g.*, the blackout on Aug 14th, 2003).

In contrast, the CPU utilization of the hierarchical control solution is controlled to stay around the schedulability bound, as shown in Figure 6(a). This is because the accuracy controller and the budget arbitrator can allocate the total utilization budget between the long subtask and the short subtasks in response to variations. In addition, both the long task controller and the short task controller can effectively control the CPU utilization to approach their respective allocated budgets as shown in Figure 4, so that the total utilization bound is enforced. Accordingly, the power grid computing system misses no deadlines and blackouts can be avoided.

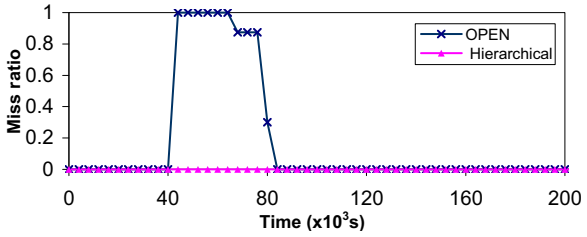
6.3 Comparison with EUCON

In this subsection, we compare the hierarchical solution with an additional baseline: EUCON.

EUCON relies on a single control algorithm to control all end-to-end tasks in the system. The control period of EUCON is selected so as to include multiple instances of the task with the longest period. However, the real-time tasks in the power grid computing system have periods with very different timescales. As a result, EUCON takes unacceptably long time to respond to any variations of the short

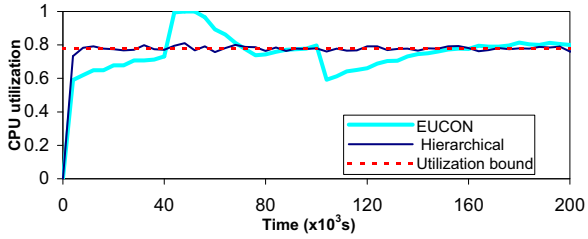


(a) Total utilization (P_1)

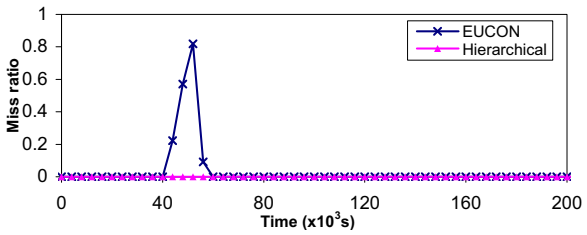


(b) Miss ratio (Task RTO)

Figure 6. Comparison with OPEN



(a) Total utilization (P_1)



(b) Miss ratio (Task RTO)

Figure 7. Comparison with EUCON

subtasks. One advantage of the hierarchical control solution over EUCON is that the hierarchical control solution uses different controllers for tasks with periods in different timescales (*i.e.*, the long utilization controller and the short task utilization controller) so that it can quickly respond to any variations of the short subtasks.

To highlight the advantage, we run experiments to simulate a scenario common to typical power grid computing systems, in which the execution time of one of the short subtasks increases dramatically at runtime due to accidental reasons. For example, in the blackout on Aug 14th, 2003, the execution time of AP abruptly increased. In this experiment, the period of EUCON is set to the same as the period of the accuracy controller (*i.e.*, $4,000s$) to include at least 2 instances of the task with the longest period (*i.e.*, STO). We also assume that the periods of the two long tasks (*i.e.*, RTO and STO) can be adjusted within a certain range. Figures 7(a) and (b) show the total utilization of the processor which accommodates one of the subtasks of AP (*i.e.*, P_1) and the miss ratio of the task of RTO, respectively. The execution time of both of the subtasks of AP increases to 2.5 times of their estimated execution time from $4,000s$ to $100,000s$. As shown in Figure 7(a), the abrupt increase leads to over-utilization of the system controlled by EUCON. EUCON takes approximately $20,000s$ (5 control periods which are around 5.5 hours) to respond to the abrupt variation by adjusting the periods of all tasks due to its long control period. Subsequently, the long duration of over-utilization leads to continuous deadline misses for the task of RTO as shown in Figure 7(b). If there is a contingency during this time, the power grid will be vulnerable to additional contingencies and hence, there is a high probability of a blackout due to delayed corrective actions.

In contrast to the slow response of EUCON, the short task utilization controller in the hierarchical control responds to the sudden variation quickly within approximately $500s$. This is because the short task utilization controller has a period much shorter than EUCON and quickly responds to the variation by adjusting the periods of all short tasks in the system. In addition, the total CPU utilization sampled with the same period of EUCON is only minimally affected by this abrupt increase as shown in Figure 8. As a result, the task of RTO has no deadline misses shown in Figure 7(b). Note that this abrupt variation may temporarily lead to several deadline misses for the short tasks which have subtasks on the same processor that AP is has subtasks running on, due to a short duration violation of the total utilization bound. However, these few deadline misses can be tolerated by most power grid computing systems in that effective corrective actions are taken timely when a contingency occurs. Therefore, no blackouts should happen due to this variation.

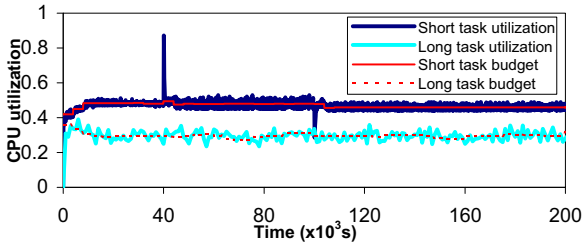


Figure 8. Utilization for short tasks and long tasks in the hierarchical control solution

7 Related Work

Most existing solutions to power grid computing are similar to OPEN that we introduced in Section 6. The research effort in the area of power grid computing primarily focuses on algorithms and models used in power grid computing tasks by reducing the execution time while enhanc-

ing accuracy. Li et al. [12] present a vision to better utilize the real-time information through the next generation monitoring and communication technology to perform a much faster online security analysis. [9] and [11] propose improved algorithms for state estimation and load forecasting, respectively. This work is different from them because we present a solution to guarantee the timeliness of power grid computing tasks despite unexpected variations by applying real time scheduling theory from a system perspective.

Research on real-time scheduling is directly related to this paper. [13] and [5] introduced a scheduling algorithm called Deferrable Server for aperiodic and sporadic jobs. This algorithm cannot be directly applied to solve our problem because the real-time tasks in power grid computing are all periodic. Abdelzaher et al. [14] and Liu et al. [15] derived a schedulibility test at the admission time for any fixed-priority scheduling algorithms to guarantee deadlines. A Feedback Control real-time Scheduling (FCS) framework [16] was proposed to provide performance guarantees for real-time systems with unknown task execution times. Wang et al. presented a distributed [17] MIMO controller to guarantee the schedulability utilization bound on multiple processors. However, all the aforementioned papers either focused on the performance of a single processor or assumed that all the tasks under control have periods and execution times approximately in the same timescale. They cannot be applied in real-time power grid computing since: 1) the subtasks of those power grid algorithms are distributed among different processors; and 2) those tasks have significantly different timescales for both periods and execution times.

Control theory has been applied in a number of other computing systems, *e.g.*, data services [18][19], power management [20] [21], and Internet servers [22][23]. A survey of feedback performance control in various computing systems is presented in [24]. This paper varies from these control-based solutions in that it aims, specifically, to support real-time power grid computing.

8 Conclusion

The timeliness of power grid computing must be strictly guaranteed to avoid disastrous blackouts. In this paper, we have proposed a hierarchical control solution to guarantee the deadlines of those power grid computing tasks by designing different controllers to control real-time tasks running on different timescales. Simulation results based on a typical workload configuration from the physical power grid demonstrate that our solution can guarantee the timeliness of power grid computing, and outperforms the existing open-loop algorithm in power grid computing and a state-of-the-art feedback scheduling algorithm designed for DRE systems.

References

[1] B. Liscouski and W. Elliot, "Final report on the august 14, 2003 blackout in the united states and canada: Causes and recommendations," in *A report to U.S. Department of Energy*, 2004.

[2] G. Glanzmann and G. Andersson, "Incorporation of n-1 security into optimal power flow for facts control," in *PSCE*, 2006.

[3] "Reliability standards for the bulk electric systems of north america," http://www.nerc.com/files/Reliability_Standards_Complete_Set_1Dec08.pdf, 2008.

[4] A. J. Wood and B. F. Wollenberg, *Power Generation, Operation, and Control, the second edition*. Wiley-Interscience, 1996.

[5] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[6] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 550–561, Jun. 2005.

[7] G. F. Franklin, J. D. Powell, and M. Workman, *Digital Control of Dynamic Systems, 3rd edition*. Prentice Hall, 1997.

[8] H. Xue, Q. quan Jia, N. Wang, Z. qian Bo, H. tang Wang, and H. xia Ma, "A dynamic state estimation method with pmu and scada measurement for power systems," *IPEC*, 2007.

[9] F. Chen, X. Han, Z. Pan, and L. Han, "State estimation model and algorithm including pmu," *DRPT*, 2008.

[10] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online with online model estimation," in *ISCA*, 2009.

[11] X. Wei, "On estimation of autoregressive signals in the presence of noise," *IEEE Trans. on Circuits and Systems*, vol. 53, no. 12, Dec. 2006.

[12] F. Li, Z. Chen, L. Fan, and P. Zhang, "Toward a Self-Healing Protection and Control System," in *NAPS*, 2008.

[13] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodicresponsiveness in hard real-time environments," *IEEE Trans. on Computers*, vol. 44, no. 1, Jan. 1995.

[14] T. Abdelzaher, V. Sharma, and C. Lu, "A utilization bound for aperiodic tasks and priority driven scheduling," *IEEE Trans. on Computers*, vol. 53, no. 3, pp. 334–350, 2004.

[15] X. Liu and T. Abdelzaher, "On Non-Utilization Bounds for Arbitrary Fixed Priority Policies," in *RTAS*, Apr. 2006.

[16] C. Lu, J. A. Stankovic, and S. H. Son, "Feedback control real-time scheduling: Framework, modeling and algorithms," *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, vol. 23, pp. 85–126, 2002.

[17] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "Deucon: Decentralized end-to-end utilization control for distributed real-time systems," *IEEE Trans. Parallel Distributed Systems*, vol. 18, no. 7, pp. 996–1009, 2007.

[18] M. Amirijoo, N. Chaufette, J. Hansson, S. H. Son, and S. Gunnarsson, "Generalized performance management of multi-class real-time imprecise data services," in *RTSS*, 2005.

[19] M. Chen, X. Wang, R. Gunasekaran, H. Qi, and M. Shankar, "Control-based real-time metadata matching for information dissemination," in *RTCSA*, 2008.

[20] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *HPCA*, 2008.

[21] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management," in *HPCA*, 2002.

[22] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Power-efficient response time guarantees for virtualized enterprise servers," in *RTSS*, 2008.

[23] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic, "ControlWare: A Middleware Architecture for Feedback Control of Software Performance," in *ICDCS*, Jul. 2002.

[24] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in software services," *IEEE Control Systems*, vol. 23, no. 3, Jun. 2003.