Self-Organized Resource Allocation for Minimizing Completion Time in Large-Scale Distributed Information Networks

Seokcheon Lee, Soundar Kumara, and Natarajan Gautam

***Abstract*—As information networks grow larger in size due to automation or organizational integration, it is important to provide simple decision-making mechanisms for each entity or groups of entities that will lead to desirable global performance. In this paper, we study a large-scale information network consisting of distributed software components linked together through a task flow structure and design a resource control mechanism for minimizing completion time. We define load index which represents component's workload. When resources are allocated locally proportional to the load index, the network can maximize the utilization of distributed resources and achieve optimal performance in the limit of large number of tasks. Coordinated resource allocation throughout the network emerges as a result of using the load index as global information. To clarify the obscurity of "large number of tasks" we provide a quantitative criterion for the adequacy of the proportional resource allocation for a given network. By periodically allocating resources under the framework of model predictive control, a closed-loop policy reactive to each current system state is formed. The designed resource control mechanism has several emergent properties that can be found in many self-organized systems such as social or biological systems. Though it is localized requiring almost no computation, it realizes desirable global performance adaptive to changing environments.**

***Index Terms*—Distributed information networks, resource allocation, completion time, adaptivity, scalability.**

I. Introduction

Critical infrastructures are increasingly becoming dependent on networked systems in many domains due to automation or organizational integration. The growth in complexity and size of software systems is leading to the increasing importance of component-based architecture [1][2]. A component is a reusable program element and component technology utilizes the components so that developers can build systems needed by simply defining their specific roles and wiring them together. In networks with component-based architecture, each component is highly specialized for specific tasks. We study a large-scale information network (with respect to the number of components as well as machines) comprising of distributed software components linked together through a task flow structure. A problem given to the network is decomposed in terms of *root tasks* for some components and those tasks are propagated through a task flow structure to other components. Since a problem can be decomposed with respect to space, time, or both, a component can have multiple root tasks that can be considered independent and identical in their nature. The service provided by the network is to produce a global solution to the given problem, which is an aggregation of the partial solutions of individual tasks. Quality of Service (QoS) of the network is determined by the time for generating the global solution, i.e. *completion time*.

For a given topology, components are sharing resources and the network can control its behavior through resource allocation, i.e. allocating resources of each machine to the components residing in that machine. In this paper we design a resource control mechanism of such networks for minimizing the completion time. Though similar problems exist in multiprocessor scheduling literature [3]-[11], they have limitations in addressing this novel resource control problem. They commonly consider the cases where each component only has to process one task after all of its predecessors complete their tasks. In contrast, a component in the networks under consideration processes multiple tasks in parallel with its successors or predecessors.

To address this kind of complex networks there is a need to facilitate some simple but effective control mechanisms. Many self-organized systems such as social and biological systems exhibit emergent properties. Though entities act with a simple mechanism without central authority, these systems are adaptive and desirable global performance can often be realized. The control mechanism designed in this paper has such properties, and hence it is applicable to large-scale networks working in a dynamic environment.

The organization of this paper is as follows. After discussing the motivation and related work in Section II, we formally define the resource control problem in detail in Section III. The

Manuscript received June 1, 2006. This work was supported in part by DARPA under Grant MDA 972-01-1-0038.

S. Lee is with the School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 USA (phone: 765-494-5419; fax: 765-494-1299; e-mail: lee46@purdue.edu).

S. Kumara is with the Department of Industrial and Manufacturing Engineering, Pennsylvania State University, University Park, PA 16802 USA (e-mail: skumara@psu.edu).

N. Gautam is with the Department of Industrial and Systems Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: gautam@tamu.edu).

control mechanism is designed in Sections IV and we provide a criterion for the adequacy of the designed control mechanism in Section V. Section VI presents empirical results, and conclusions and future work are discussed in Section VII.

II. Motivation and Related Work

A. UltraLog Networks

UltraLog networks (dtsn.darpa.mil/ixo/programs.asp?id=61) [12]-[17], implemented in Cougaar (Cognitive Agent Architecture: www.cougaar.org) developed by DARPA (Defense Advanced Research Project Agency), are the next-generation military logistics information systems. In Cougaar, a software system comprises of agents and an agent of components (called plugins). The task flow structure in these systems is that of components as a combination of intra-agent and inter-agent task flows. Each agent in an UltraLog network represents an organization of military supply chain, and has a set of components specialized for each functionality (allocation, expansion, aggregation, inventory management, message transport, etc) and class (ammunition, water, fuel, etc).

The objective of an UltraLog network is to produce a logistics plan for a given military operation, which is an aggregate of individual schedules built by components. An operation is transformed into logistics requirements and the requirements are decomposed into root tasks (one task per day) for designated components. As a result, a component can have hundreds of root tasks depending on the horizon of the operation and thousands of tasks as the root tasks are propagated. As the scale of operation increases, there can be thousands of agents (tens of thousands of components) in hundreds of machines working together to generate a logistics plan. The system makes initial planning and continuous replanning to cope with logistics plan deviations or operational plan changes. Initial planning and replanning are the instances of the current research problem.

One of the important performance criteria of these networks is the (plan) completion time. This metric directly affects the performance of the military operations. The question is how to manage the resources of the networks in order to minimize the completion time.

B. Scheduling

The resource control problem under consideration is a scheduling problem. In general, a scheduling problem is allocating limited resources to a set of tasks to optimize a specific objective. One widely studied objective is completion time (also called makespan) as in the problem we have considered. Though there are a variety of formulations and algorithms available in multiprocessor scheduling literature [3]-[11], they have limitations in addressing the scheduling problem under consideration. They commonly consider so-called workflow applications where each component only has to process one task after all of its predecessors complete their tasks[1]. In contrast, a component in the networks under consideration processes multiple tasks in parallel with its successors or predecessors. The critical path used to determine the completion time, is not valid in the scheduling problem we are addressing.

Though it is not easy to find a problem exactly same as ours, it is possible to convert our problem into a job shop scheduling problem. In a job shop, there are a set of jobs and a set of machines. Each job has a set of serial operations and each operation should be processed on a specific machine. A job shop scheduling problem is sequencing the operations in each machine by satisfying a set of job precedence constraints such that the completion time is minimized. Our problem can be transformed into the job shop scheduling problem. However, job shop scheduling problems are in general intractable. Though the job shop scheduling problem is polynomially solvable when there are two machines and each job has two operations, it becomes NP-hard on the number of jobs even if the number of machines or operations is more than two [18][19]. Considering that the task flow structure of our networks is arbitrary, our scheduling problem is NP-hard on the number of components in general. The increase of the number of tasks makes the problem even harder. To address this complex scheduling problem there is a need to facilitate some simple but effective scheduling algorithms.

III. Problem Specification

In this section we formally define the problem in a general form by detailing network model and resource allocation. We concentrate on computational CPU resources assuming that the system is computation-bounded. (The notations used repetitively throughout the paper are summarized in Appendix.)

A. Network Model

The network is composed of a set $\boldsymbol{A}$ of components and a set $\boldsymbol{N}$ of machines. $\boldsymbol{K_n}$ denotes a set of components that reside in machine n sharing the machine's CPU resource. Task flow structure of the network, which defines precedence relationship between components, is an arbitrary directed acyclic graph. A problem given to the network is decomposed in terms of *root tasks* for some components and those tasks are propagated through the task flow structure. Each component processes one of the tasks in its queue (which has root tasks as well as tasks from predecessor components) and then sends it to successor components. We denote the number of root tasks and expected CPU time[2] per task of component i as $<rt_i, P_i>$ respectively. Fig. 1 shows an example network in which there are four components residing in three machines. Components A_1 and A_2 resides in N_1 and each of them has 100 root tasks. A_3 in N_2 and

[1] We will discuss in Section VII how the work in this paper can be applied to the workflow applications when there are multiple jobs to be processed in batch.

[2] The distribution of CPU time can be arbitrary though we use only expected CPU time in the control mechanism.

A_4 in N_3 have no root tasks, but each of them has 100 tasks from the corresponding predecessors, namely A_1 and A_2 respectively.

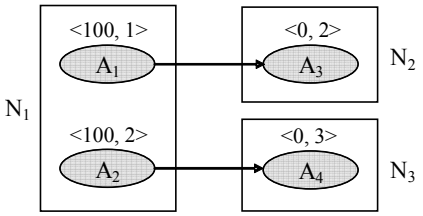


Fig. 1. An example network composed of four components in three machines. Components A_1 and A_2 reside in N_1 and each of them has 100 root tasks. A_3 in N_2 and A_4 in N_3 have no root tasks, but each of them has 100 tasks from the corresponding predecessors.

B. Resource Allocation

When there are multiple components in a machine, the network needs to control its behavior through resource allocation. In the example network, machine N_1 has two components and the system performance can depend on its resource allocation to these two components. There are several CPU scheduling algorithms for allocating a CPU resource amongst multiple threads. Among the scheduling algorithms, proportional CPU share (PS) scheduling [20]-[23] is known for its simplicity, flexibility, and fairness. In PS scheduling, threads are assigned weights and resource shares are determined proportional to the weights. Excess CPU time from some threads is allocated fairly to other threads. We adopt PS scheduling as resource allocation scheme because of its generality in addition to the benefits mentioned above. We define resource allocation variable set $\boldsymbol{w} = \{w_i(t): i \in \boldsymbol{A}, t \geq 0\}$ in which $w_i(t)$ is a non-negative weight of component i at time t. If total managed weight of a machine n is ω_n, the boundary condition for assigning weights over time can be described as:

$$\sum_{i \in \boldsymbol{K_n}} w_i(t) = \omega_n \quad where\ w_i(t) \geq 0\ . \tag{1}$$

C. Problem Definition

The service provided by the network is to produce a global solution to a given problem, which is an aggregate of partial solutions of individual tasks. QoS is determined by completion time taken to generate the global solution. In this paper we design a resource control mechanism to minimize the completion time T though resource allocation $\boldsymbol{w}$ as in (2).

$$\arg\min_{w} \ T \tag{2}$$

IV. Resource Control Mechanism Design

There are two representative optimal control approaches in dynamic systems: Dynamic Programming (DP) and Model Predictive Control (MPC). Though DP gives optimal closed-loop policy it has inefficiencies in dealing with large-scale systems especially when systems are working in finite time horizon. In MPC, for each current state, an optimal open-loop control policy is designed for finite-time horizon by solving a static mathematical programming model [24]-[26]. The design process is repeated for the next observed state feedback forming a closed-loop policy reactive to each current system state. Though MPC does not give absolute optimal policy in stochastic environments, the periodic design process alleviates the impacts of stochasticity. Considering the characteristics of the problem under consideration, we choose MPC framework. The networks are large-scale and work in finite time horizon.

Under the MPC framework, we need to build a mathematical programming model which is essentially a scheduling problem formulation. However, we pursue directly an optimal resource allocation policy without explicit formulation of the mathematical programming model. One important characteristic of the networks under consideration is that each component processes tasks in parallel with its predecessors or successors. In this section, we investigate the impacts of the parallelism on the optimal resource allocation policy in the limit of large number of tasks. For theoretical analysis, we assume a hypothetical weighted round-robin server for CPU scheduling though it is not strictly required in practice. The hypothetical server has idealized fairness as the CPU time received by each thread in a round is infinitesimal and proportional to the weight of the thread. But, the arguments we will make do not seem to be invalid because they are based on worst-case analysis and quantum size is relatively infinitesimal compared to working horizon in reality.

A. Effects of Resource Allocation

The completion time T is the time taken to generate the global solution, i.e., to process all the tasks of a network. Let T_n and T_i be the completion times of machine n and component i. Then, the relationships as in (3) hold.

$$T = \underset{n \in \boldsymbol{N}}{Max}\ T_n = \underset{i \in \boldsymbol{A}}{Max}\ T_i\ , \quad T_n = \underset{i \in \boldsymbol{K_n}}{Max}\ T_i\ . \tag{3}$$

A component's instantaneous resource availability $RA_i(t)$ is the available fraction of a resource when the component requests the resource at time t. Service time $S_i(t)$ is the time taken to process a task at time t and has a relationship with $RA_i(t)$ as:

$$\int_t^{t+S_i(t)} RA_i(\tau) d\tau = P_i\ . \tag{4}$$

When $RA_i(t)$ remains constant, $S_i(t)$ becomes:

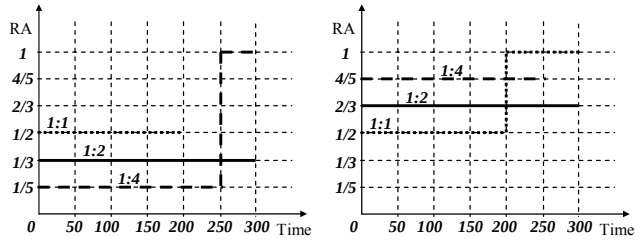
$$S_i(t) = \frac{P_i}{RA_i(t)}\ . \tag{5}$$

Now, consider the example network in Fig. 1. In the network only N_1 has the chance to allocate its resource since it has two residing components. T_{N1} is invariant to resource allocation and equal to 300 (=100*1+100*2). But, T_{A1} and T_{A2} can vary

depending on the resource allocation of N_1. When the resource is allocated equally to the components, both $RA_{A1}(t)$ and $RA_{A2}(t)$ are equal to 0.5 initially. As A_1 completes at t=200 (=100*1/0.5), A_2 starts utilizing the resource fully from then, i.e. $RA_{A2}(t)$=1 for $t \geq 200$. So, A_2 completes 50 tasks at t=200 (=50*2/0.5) and remaining 50 tasks at t=300 (=200+50*2/1). A_3 completes at t=202 (=200+1*2/1) because task inter-arrival time from A_1 is equal to its service time. As A_4's service time is less than task inter-arrival time (=4) for $t \leq 200$, A_4 completes 49 tasks at t=200 with one task in queue arriving at t=200. From t=200, task inter-arrival time from A_2 becomes reduced to 2 which is less than A_4's service time. So, tasks become accumulated till t=300 and A_4 completes at t=353 (=200+51*3/1). In this way we trace exact system behavior under three resource allocation strategies as shown in Fig. 2.

	$w_{A1} : w_{A2}$		
	1 : 1	1 : 2	1 : 4
T_{A1}	200	300	300
T_{A2}	300	300	250
T_{A3}	202	302	352
T_{A4}	353	303	302.5
T	353	303	352

(a) Completion time



(b) Resource availability of A_1 *(c) Resource availability of A_2*

Fig. 2. Effects of resource allocation. Depending on the resource allocation of machine N_1, each of components A_1 and A_2 follows different resource availability profile as in (b) and (c). Consequently, the differences result in different completion times as in (a).

The network cannot complete at less than t=300 because each of N_1 and N_3 requires 300 CPU time. When the resource is allocated with 1:2 ratio, the completion time T is minimal close to 300. The ratio is proportional to each component's total required CPU time, i.e., 1:2 ≡ 100*1:100*2. One interesting question is whether the proportional allocation can give the best performance regardless of the parameters of A_3 and A_4. The answer is yes. If a component A_1 is allocated more resource than the proportional allocation, T_{A3} is dominated by the maximal of T_{A1} and A_3's total CPU time. But, the first quantity is less than T_{N1} and the second quantity is an invariant. So, allocating more resource than the proportional allocation cannot help reducing the completion time of the network. However, if a component is allocated less resource than the proportional allocation, its successor's task inter-arrival time is stepwise decreasing. As a result, the successor underutilizes resources and can increase the completion time of the network. Therefore, the proportional allocation leads the network to efficiently utilize distributed resources and consequently helps minimizing the completion time of the network, though it is localized independent of the successors' parameters.

B. Optimal Resource Allocation

To generalize the arguments for arbitrary network configurations, we define *Load Index* LI_i which represents component i's total CPU time required to process its tasks. As a component needs to process its own root tasks as well as incoming tasks from its predecessors, its number of tasks L_i is identified as in (6) where $\underline{\boldsymbol{i}}$ denotes the immediate predecessors of component i. Then, LI_i is represented as in (7).

$$L_i = rt_i + \sum_{a \in \underline{\boldsymbol{i}}} L_a \tag{6}$$

$$LI_i = L_i P_i \tag{7}$$

Also, we define a term called *task availability* as an indicator of relative preference for task arrival patterns. A component's task availability for an arrival pattern is higher than for another if cumulative number of arrived tasks is larger than or equal to over time. A component prefers a task arrival pattern with higher task availability since it can utilize more resources. Consider a network and reconfigure it such that all components have their tasks in their queues at t=0. Each component has maximal task availability in the reconfigured network and the completion time of the reconfigured network forms the lower bound T^{LB} of a network's completion time T given by:

$$T^{LB} = \underset{n \in \boldsymbol{N}}{Max} \sum_{i \in \boldsymbol{K_n}} LI_i \ . \tag{8}$$

Now, convert a network into a network with infinitesimal tasks. Each root task is divided into r infinitesimal tasks and each P_i is replaced with P_i/r. Then, the load index of each component is the same as the original network but tasks are infinitesimal. We denote the completion time of the network with infinitesimal tasks as T'.

Theorem 1. *T' equals to T^{LB} when each machine allocates its resource proportional to its residing components' load indices as:*

$$w_i(t) = w_i = \frac{LI_i}{\sum_{p \in \boldsymbol{K_{n(i)}}} LI_p} \omega_{n(i)} \quad \text{for all } t \geq 0 \ , \tag{9}$$

where n(i) denotes a machine in which component i resides.

Proof. Instantaneous resource availability $RA_i(t)$ is more[3] than or equal to assigned weight proportion as:

[3] because excess resources of some components are utilized by others in proportional CPU share scheduling.

$$RA_i(t) \geq \frac{w_i(t)}{\omega_{n(i)}} \quad for\ t \geq 0\ . \tag{10}$$

Then, under the proportional resource allocation policy, service time $S_i(t)$ is less than or equal to T^{LB}/L_i over time as shown in (11). Therefore, any component can complete at less than or equal to T^{LB} if it receives infinitesimal tasks at a constant interval or more preferably.

$$\begin{aligned} P_i &= \int_t^{t+S_i(t)} RA_i(\tau)d\tau \geq \int_t^{t+S_i(t)} \frac{w_i(\tau)}{\omega_{n(i)}} d\tau \\ &= \frac{w_i}{\omega_{n(i)}} S_i(t) = \frac{LI_i}{\sum_{p \in \mathbf{K_{n(i)}}} LI_p} S_i(t) \geq \frac{LI_i}{T^{LB}} S_i(t) \\ &\Rightarrow \frac{T^{LB}}{L_i} \geq S_i(t) \quad for\ t \geq 0 \end{aligned} \tag{11}$$

The components with no predecessors will generate infinitesimal tasks at a constant interval or more preferably in $0 \leq t \leq T^{LB}$ since they have all the tasks in their queues at $t=0$. Consequently, all the components will receive and generate tasks at a constant interval in $0 \leq t \leq T^{LB}$ or more preferably. Therefore, the network completes at T^{LB} under the proportional allocation policy. □

From Theorem 1 we can conjecture that a network can achieve a performance close to T^{LB} under proportional allocation in the limit of large number of tasks. We propose the proportional allocation as an optimal resource allocation policy. Though the proportional allocation is *localized*, the network can maximize the utilization of distributed resources and achieve desirable performance. Coordinated resource allocation throughout the network emerges as a result of using the load index as global information. If machines do not follow the proportional allocation policy, some components can receive their tasks less preferably resulting in underutilization and consequently increased completion time as have shown in the previous subsection.

Another important property of the proportional allocation policy is that it is itself *adaptive*. Suppose there are some stressors sharing resources together with the components. Let ω_n^s be the amount of shared resources by a stressor in machine n. Then, the lower bound performance T_s^{LB} under stress is given by (12). We denote the completion time under stress as T_s'.

$$T_s^{LB} = \underset{n \in \mathbf{N}}{Max} \frac{\omega_n + \omega_n^s}{\omega_n} \sum_{i \in \mathbf{K}_n} LI_i \tag{12}$$

Theorem 2. *T_s' equals to T_s^{LB} under proportional allocation.*

Proof. $RA_i(t)$ becomes:

$$RA_i(t) \geq \frac{w_i(t)}{\omega_{n(i)} + \omega_{n(i)}^s} \quad for\ t \geq 0\ . \tag{13}$$

Then, (11) results in (14) under proportional allocation.

$$\frac{T_s^{LB}}{L_i} \geq S_i(t) \quad for\ t \geq 0 \tag{14}$$

Therefore, the network completes at T_s^{LB} under proportional allocation. □

Theorem 2 depicts that the proportional allocation policy is optimal independent of the stress environments. Though we do not consider the environments explicitly, the policy gives lower bound performance adaptively. This characteristic is especially important when the system is vulnerable to unpredictable stress environments. Modern networked systems can be easily exposed to various adverse events such as accidental failures and malicious attacks, and the space of stress environment is high-dimensional and also evolving [27]-[29].

C. Resource Control Mechanism

Consider current time as t. To update load index as the system moves on, we slightly modify it to represent total CPU time for the remaining tasks as:

$$LI_i(t) = R_i(t) + L_i(t)P_i\ , \tag{15}$$

in which $R_i(t)$ denotes remaining CPU time for a task in process and $L_i(t)$ the number of remaining tasks excluding a task in process. After identifying initial number of tasks $L_i(0)=L_i$, each component updates it by counting down as they process tasks. Following the MPC framework, a resource manager of each machine periodically collects current $LI_i(t)$ from residing components and allocates resources proportional to the indices as in (16). Since the resource allocation policy is purely localized there is no need for synchronization between machines.

$$w_i(t) = \frac{LI_i(t)}{\sum_{p \in \mathbf{K_{n(i)}}} LI_p(t)} \omega_{n(i)} \tag{16}$$

V. Adequacy Criterion

There are several desirable properties of the proportional resource allocation policy. Though it is localized requiring almost no computation, it realizes desirable global performance adaptive to changing environments. However, though such properties hold in the limit of large number of tasks, such largeness does not hold in reality. So, we define a general criterion by which one can evaluate if the properties will hold for a given network. For this purpose we characterize upper bound performance of a network under proportional allocation.

Theorem 3. *Under proportional allocation, a network's upper bound T^{UB} of completion time T is given by:*

$$T^{UB} = T^{LB} + \underset{e\in \boldsymbol{E}}{Max}\ \underset{\boldsymbol{J}\in \boldsymbol{S_e}}{Max} \sum_{i\in \boldsymbol{J}} \lceil P_i \sum_{p\in \boldsymbol{K_{n(i)}}} LI_p / LI_i \rceil , \quad (17)$$

where $\boldsymbol{E}$ denotes a set of components which have no successor and $\boldsymbol{S_e}$ a set of task paths to component e. A task path to component e is a set of components in a path from a component with no predecessor to component e, and does not include component e.

Proof. From (11) we can induce the lowest upper bound S_i^{UB} of $S_i(t)$ as:

$$S_i^{UB} = P_i \sum_{p\in \boldsymbol{K_{n(i)}}} LI_p / LI_i . \quad (18)$$

So, a component $e\in\boldsymbol{E}$ (with no successors) can receive tasks at a constant interval of T^{LB}/L_e from maximal task traveling time to the component as in (19) in the worst case. Consequently, the component e can complete at least at (20) and the upper bound T^{UB} is the maximal of the bounds.

$$\underset{\boldsymbol{J}\in \boldsymbol{S_e}}{Max} \sum_{i\in \boldsymbol{J}} S_i^{UB} \quad (19)$$

$$T^{LB} + \underset{\boldsymbol{J}\in \boldsymbol{S_e}}{Max} \sum_{i\in \boldsymbol{J}} S_i^{UB} \qquad \square \quad (20)$$

So, a network can achieve a performance in $T^{LB} \le T \le T^{UB}$ under the proportional resource allocation policy. We define *adequacy criterion* as in (21) which is the ratio between T^{LB} and T^{UB}. If this criterion is close to one, i.e. the upper bound is close to the lower bound, one can ensure the optimality of the proportional resource allocation policy. The example network in Fig. 1 is quite adequate because the adequacy criterion is 0.99 (300/303). Note that the adequacy can be high even though the number of tasks for some components is small.

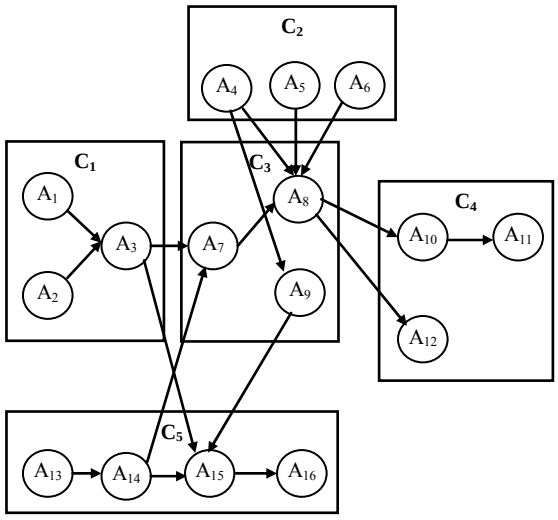
$$Adequacy = \frac{T^{LB}}{T^{UB}} \quad (21)$$

VI. Empirical Results

We ran several experiments using a discrete-event simulator to validate the resource control mechanism designed.

A. Experimental Design

We consider a network presented in [33], which is a small-size UltraLog network discussed earlier. The network is composed of sixteen components in five clusters, C_1, C_2, C_3, C_4, and C_5 as shown in Fig. 3. The components in a cluster are not separable to different machines. There are three machines available $\boldsymbol{N} = \{N_1, N_2, N_3\}$ with ω_n=1 for all $n\in\boldsymbol{N}$. We consider two network topologies: Top-A and Top-B. In Top-A, $\boldsymbol{K_{N1}} = \{C_1, C_2\}$, $\boldsymbol{K_{N2}} = \{C_3, C_5\}$, and $\boldsymbol{K_{N3}} = \{C_4\}$. In Top-B, $\boldsymbol{K_{N1}} = \{C_1, C_5\}$, $\boldsymbol{K_{N2}} = \{C_2, C_3\}$, and $\boldsymbol{K_{N3}} = \{C_4\}$. To implement proportional CPU share scheduling we use a weighted round-robin server in which CPU time received by each component in a round is equal to its assigned weight. If a component's queue becomes empty before utilizing all of the received CPU time, the server goes to the next component immediately.



Top-A: {C_1, C_2} {C_3, C_5} {C_4}
Top-B: {C_1, C_5} {C_2, C_3} {C_4}

Fig. 3. Experimental network composed of sixteen components in five clusters. There are three machines and two network topologies, Top-A and Top-B, are considered.

The components without predecessors, A_1, A_2, A_4, A_5, A_6, and A_{13}, are assigned equal number NRT of root tasks, i.e. rt_i = NRT for $i \in \{A_1, A_2, A_4, A_5, A_6, A_{13}\}$ and $rt_i = 0$ otherwise. To observe the impact of the number of tasks, we vary the NRT with 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, and 2^9. CPU time per task P_i is equal to 1 for all $i \in \boldsymbol{A}$, and the distribution of P_i can be deterministic or exponentially distributed. While using a stochastic distribution we repeat 30 experiments.

Two resource control policies are used for each experimental condition. Table II shows these control policies. In round-robin allocation policy (RR) the components in each machine are assigned equal weights over time. In contrast, proportional allocation policy (PA) allocates resources proportional to the load indices periodically (every 100 time units).

TABLE II
Control Policies for Experimentation

Control policy	Description
RR	Round-Robin allocation
PA	Proportional allocation

B. Numerical Results

Numerical results from the experimentation are shown in Fig. 4. For each NRT in each topology, lower bound T^{LB} and upper bound T^{UB} are computed from (8) and (17). The bounds are used to compute the adequacy criterion described in (21). The adequacy tends to increase with the increase of the number of tasks. So, the network under PA policy can achieve close performance to T^{LB} when the number of tasks is large. The completion times under RR and PA policies in deterministic and stochastic environments are represented in time units as well as in ratios to T^{LB}.

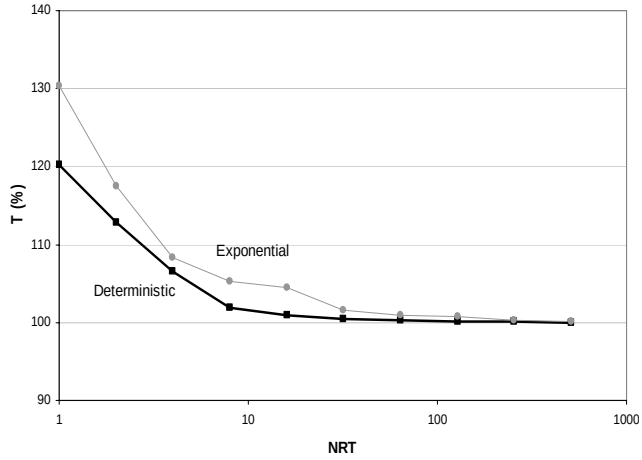
As one can observe, PA policy outperforms RR policy in all different conditions, supporting the effectiveness of the proportional resource allocation policy. The network under RR policy is considered as the one without control and the implementation of the control with PA policy improves the performance of the network. Also, the performance under PA policy is almost same as T^{LB} when the number of tasks is large in all different conditions, validating the argument we have made. We argued that the network under PA policy can achieve T^{LB} in the limit of large number of tasks.

There exist large gaps between the completion time and T^{LB} when NRT is small. However, notice that the performance rapidly converges into T^{LB} with the increase of the number of tasks, as shown in Fig. 4(a) for Top-A and Fig 4(b) for Top-B (note that the scale of x-axis is logarithmic). This observation indicates that the PA policy can be beneficial even though the network has low adequacy with relatively small number of tasks.

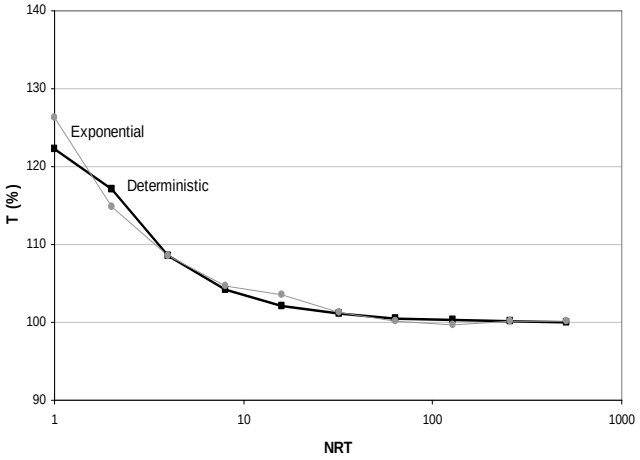
					Deterministic				Exponential			
					RR		PA		RR		PA	
Topology	NRT	T^{LB}	T^{UB}	Adequacy (%)	T	T/T^{LB} (%)	T	T/T^{LB} (%)	T	T/T^{LB} (%)	T	T/T^{LB} (%)
Top-A	2^0	20.00	73.00	27.40	27.00	135.00	24.05	120.25	28.53	142.67	26.06	130.30
	2^1	40.00	93.00	43.01	51.00	127.50	45.10	112.75	52.26	130.64	46.97	117.42
	2^2	80.00	133.00	60.15	99.00	123.75	85.30	106.63	98.17	122.71	86.69	108.36
	2^3	160.00	213.00	75.12	195.00	121.88	163.00	101.88	196.00	122.50	168.54	105.34
	2^4	320.00	373.00	85.79	387.00	120.94	323.00	100.94	386.29	120.72	334.31	104.47
	2^5	640.00	693.00	92.35	771.00	120.47	643.00	100.47	765.20	119.56	649.65	101.51
	2^6	1280.00	1333.00	96.02	1539.00	120.23	1283.00	100.23	1536.16	120.01	1291.30	100.88
	2^7	2560.00	2613.00	97.97	3075.00	120.12	2563.00	100.12	3086.25	120.56	2580.44	100.80
	2^8	5120.00	5173.00	98.98	6147.00	120.06	5123.00	100.06	6163.53	120.38	5131.10	100.22
	2^9	10240.00	10293.00	99.49	12291.00	120.03	10243.00	100.03	12325.30	120.36	10247.50	100.07
Top-B	2^0	18.00	55.50	32.43	23.00	127.78	22.00	122.22	24.36	135.32	22.74	126.34
	2^1	36.00	73.50	48.98	44.00	122.22	42.15	117.09	44.75	124.31	41.32	114.77
	2^2	72.00	109.50	65.75	86.00	119.44	78.15	108.55	85.65	118.96	78.12	108.50
	2^3	144.00	181.50	79.34	170.00	118.06	150.15	104.27	169.99	118.05	150.70	104.65
	2^4	288.00	325.50	88.48	338.00	117.36	294.15	102.14	338.86	117.66	298.09	103.50
	2^5	576.00	613.50	93.89	674.00	117.01	582.15	101.07	666.59	115.73	582.98	101.21
	2^6	1152.00	1189.50	96.85	1346.00	116.84	1158.10	100.53	1337.48	116.10	1154.47	100.21
	2^7	2304.00	2341.50	98.40	2690.00	116.75	2310.10	100.26	2675.55	116.13	2297.95	99.74
	2^8	4608.00	4645.50	99.19	5378.00	116.71	4614.10	100.13	5362.45	116.37	4611.99	100.09
	2^9	9216.00	9253.50	99.59	10754.00	116.69	9222.10	100.07	10748.85	116.63	9236.41	100.22

NRT: number of root tasks, RR: Round-Robin allocation, PA: Proportional allocation

(a) Numerical results



(b) Top-A



(c) Top-B

Fig. 4. Experimental results: (a) Completion times in different experimental conditions; (b) Completion time as a function of NRT in topology Top-A, represented in ratio to T^{LB}; (c) Completion time as a function of NRT in topology Top-B, represented in ratio to T^{LB}.

VII. Conclusions

We designed a simple but effective resource control mechanism for the novel information networks composed of large number of distributed software components. The resource control mechanism has several desirable properties. First, it is localized as each machine can make decisions independent of others. Second, it requires almost no computation. Third, nevertheless the network can achieve a desirable performance. Fourth, it is itself adaptive to the stress environments without explicit considerations. Such emergent properties can be found in many self-organized systems such as social or biological systems. Though entities act with a simple mechanism without central authority, desirable global performance can often be realized. When a large-scale network is working in a dynamic environment under the control mechanism, it is really a self-organized system.

It would be valuable to investigate the impact of control period. The designed resource control mechanism requires periodic updating of load indices and weights. With reduced control period, one can expect better performance in stochastic environments because of the alleviation of the impacts of stochasticity. Fig. 5 shows the performance of the network as a function of control period with NRT = 50 in topology Top-B. The performance tends to improve with the decrease of the control period, though the improvement is less than 1%. However, the reduction of the control period will lead to consuming more resources for the control purposes, resulting in increased completion time. Therefore, there exists some tradeoff in the optimality of the control periods. To find optimal control period, we need to reinforce the prediction model with the control period incorporated.

The work in this paper can also be extended to the network topology problem. A network topology problem assigns components to available machines with a set of constraints. Some components may not be separable to different machines and may be allowed to specific machines. Though the network topology problem is intertwined with the resource allocation problem in general, the proportional allocation policy we have addressed completely separates them. Under the proportional allocation policy, the performance of a topology can be estimated by the maximal of total CPU time of each machine. Therefore, the network topology problem becomes the easiest multiprocessor scheduling problem, i.e., an assignment of independent clusters (cluster: a set of inseparable components) to machines. The topology Top-B in the experimentation is the optimal one which minimizes the completion time. The optimal topology significantly outperforms non-optimal one Top-A by about 10%.

This topology problem is known as NP-complete [30][31] and there are diverse heuristic algorithms available in the literature. Eleven heuristics were selected and examined with various problem configurations in [32]. They are Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min-min, Max-min, Duplex, Genetic Algorithm, Simulated Annealing, Genetic Simulated Annealing, Tabu, and A*. Though Genetic Algorithm always gave the best performance, if algorithm execution time is also considered, it was shown that the simple Min-min heuristic performs well in comparison to others.

The workflow applications in [3]-[11] can be addressed using the work in this paper, when they process multiple jobs in batch. Suppose there are m jobs assigned to a workflow application. Each component without predecessors will have m root tasks, whereas each component with predecessors will have no root tasks but it has m tasks to process incoming from its predecessors. When a component has multiple predecessors, it processes k^{th} task only after all the k^{th} tasks of its predecessors arrive. For example, when we consider the example network in Fig. 1 from the viewpoint of workflow applications, there are 100 jobs in batch and every component has 100 tasks to process. The optimality of the proportional allocation policy also holds when the number of jobs in batch is large, and hence the resource control mechanism designed in this paper can be directly used to determine resource allocation and consequently network topology of the workflow applications. Though the workflow research focuses on a single job, there will be some situations where batch processing is more desirable. For example, one may want to process multiple images altogether because they are available at the same time.

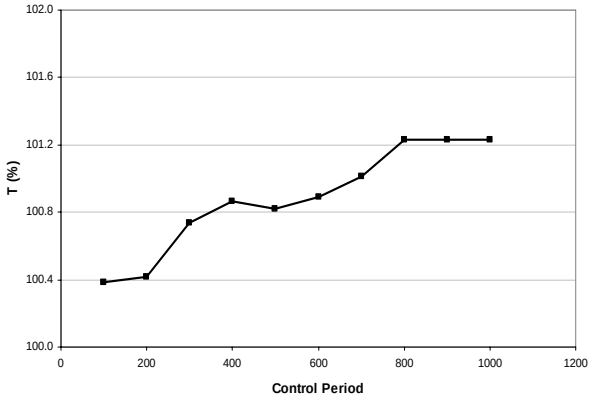


Fig. 5. Completion time as a function of control period with NRT = 50 in topology Top-B.

Appendix. Notations

Sets
$\boldsymbol{A}$: a set of components
$\boldsymbol{N}$: a set of machines
$\boldsymbol{K_n}$: a set of components in machine n
$\underline{\boldsymbol{i}}$: immediate predecessors of component i
$n(i)$: a machine in which component i resides

Variables
$w_i(t)$: weight of component i at time t
T: completion time

Parameters
rt_i: number of root tasks of component i
P_i: expected CPU time per task of component i
ω_n: total managed weight of machine n
T^{LB}: lower bound of completion time T

T^{UB}: upper bound of completion time T under proportional resource allocation
T_s^{LB}: lower bound of completion time T under stress
$RA_i(t)$: instantaneous resource availability of component i at time t
$R_i(t)$: remaining CPU time for a task in process of component i at time t
$L_i(t)$: number of remaining tasks of component i at time t, $L_i(0)=L_i$

Functions
$LI_i(t)$: load index, component i's total CPU time required at time t, $LI_i(0)=LI_i$

REFERENCES

[1] B. Meyer, "On to components," *IEEE Computer*, vol. 32, no. 1, pp. 139-140, 1999.

[2] P. Clements, "From subroutine to subsystems: component-based software development," in *Component Based Software Engineering*, A. W. Brown, Ed. IEEE Computer Society Press, 1996, pp. 3-6.

[3] Y. K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, 406-471, 1999.

[4] V. S. Adve and M. K. Vernon, "Parallel program performance prediction using deterministic task graph analysis," *ACM Transactions on Computer Systems*, vol. 22, no. 1, 94-136, 2004.

[5] T. Hagras and J. Janecek, "A fast compile-time task scheduling heuristic for homogeneous computing environments," *Int. J. of Computers and Their Applications*, vol. 12, no. 2, 76-82, 2005.

[6] T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," *Parallel Computing*, vol. 31, no. 7, 653-670, 2005.

[7] M. M. Eshaghian and Y. C. Wu, "Mapping heterogeneous task graphs onto heterogeneous system graphs," in *Proc. 6th Heterogeneous Computing Workshop*, 1997, pp. 147-161.

[8] H. El-Rewini, H. H. Ali, and T. Lewis, "Task scheduling in multiprocessor systems," *Computer*, vol. 28, no. 12, 27-37, 1995.

[9] D. C. Li and N. Ishii, "Scheduling task graphs onto heterogeneous multiprocessors," in *Proc. IEEE Region 10's Ninth Annual International Conference*, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 556-563.

[10] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *SIGMOD Record*, vol. 34, no. 3, pp. 44-49, 2005.

[11] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: state of the art and open problems," School of Computing, Queen's University, Kingston, Ontario, Tech. Rep. 2006–504, Jan. 2006.

[12] M. Brinn, J. Berliner, A. Helsinger, T. Wright, M. Dyson, S. Rho, and D. Wells, "Extending the limits of DMAS survivability: the UltraLog Project," *IEEE Intelligent Systems*, vol. 19, no. 5, pp. 53-61, 2004.

[13] D. Moore, W. Wright, and R. Kilmer, "Control surfaces for Cougaar," in *Proc. First Open Cougaar Conference*, 2004, pp. 37-44.

[14] W. Peng, V. Manikonda, and S. Kumara, "Understanding agent societies using distributed monitoring and profiling," in *Proc. First Open Cougaar Conference*, 2004, pp. 53-60.

[15] H. Gupta, Y. Hong, H. P. Thadakamalla, V. Manikonda, S. Kumara, and W. Peng, "Using predictors to improve the robustness of multi-agent systems: design and implementation in Cougaar," in *Proc. First Open Cougaar Conference*, 2004, pp. 81-88.

[16] D. Moore, A. Helsinger, and D. Wells, "Deconfliction in ultra-large MAS: issues and a potential architecture," in *Proc. First Open Cougaar Conference*, 2004, pp. 125-133.

[17] R. D. Snyder and D. C. Mackenzie, "Cougaar agent communities," in *Proc. First Open Cougaar Conference*, 2004, pp. 143-147.

[18] T. Gonzalez and S. Sahni, "Flowshop and jobshop schedules: complexity and approximation," *Operations Research*, vol. 26, pp. 36-52, 1978.

[19] J. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343-362, 1977.

[20] I. Stoica, H. Abdel-Wahab, J. Gehrke, K. Jeffay, S. K. Baruah, and C. G. Plexton, "A proportional share resource allocation algorithm for real-time, time-shared systems," in *Proc. 17th IEEE Real-Time Systems Symposium*, 1996, pp. 288-299.

[21] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Internetworking Research and Experience*, vol. 1, pp. 3-26, 1990.

[22] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: flexible proportional-share resource management," in *Proc. First Symposium on Operating System Design and Implementation*, 1994, pp. 1-11.

[23] B. Caprita, W. C. Chan, J. Nieh, C. Stein, and H. Zheng, "Group ratio round-robin: O(1) proportional share scheduling for uniprocessor and multiprocessor systems," in *Proc. 2005 USENIX Annual Technical Conference, Anaheim*, CA, 2005.

[24] J. B. Rawlings, "Tutorial overview of model predictive control," *IEEE Control Systems*, vol. 20, no. 3, pp. 38-52, 2000.

[25] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers and Chemical Engineering*, vol. 23, no. 4, pp. 667-682, 1999.

[26] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive technology," *Control Engineering Practice*, vol. 11, pp. 733-764, 2003.

[27] S. Jha and J. M. Wing, "Survivability analysis of networked systems," in *Proc. 23rd Int. Conf. Software engineering*, 2001, pp. 307-317.

[28] A. P. Moore, R. J. Ellison, and R. C. Linger, "Attack modeling for information security and survivability," Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA, Tech. Note CMU/SEI-2001-TN-001, 2001.

[29] F. Moberg, "Security analysis of an information system using an attack tree-based methodology," M.S. thesis, Automation Engineering Program, Chalmers University of Technology, Sweden, 2000.

[30] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *J. of the ACM*, vol. 24, no. 2, pp. 280-289, 1977.

[31] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427-1436, 1989.

[32] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810-837, 2001.

[33] S. Lee and S. Kumara, "Estimating global stress environment by observing local behavior in a multiagent system," in *Proc. IEEE Int. Conf. Automation Science and Engineering*, 2005, pp. 215-219.

Seokcheon Lee is currently a visiting assistant professor in the School of Industrial Engineering at Purdue University. Previously, he worked as a research associate in the Industrial Engineering Department at the Pennsylvania State University after he had received his Ph.D. in the same department in 2005. He received M.S. and B.S. degrees in Industrial Engineering from the Seoul National University, South Korea, in 1993 and 1991 respectively. His research interests include distributed control methodologies from economics, swarm intelligence, learning, and complex network theory, in the areas of computer and communication networks, supply chain and manufacturing systems, and integrated systems resulting from emerging computing technologies.

Soundar Kumara is a distinguished professor in the Industrial Engineering Department at the Pennsylvania State University. He also holds joint appointments with the School of Information Sciences and Technology and the Department of Computer Science and Engineering. He received his Ph.D. in Industrial Engineering from the Purdue University and M. Tech. in Industrial Engineering from the Indian Institute of Technology, India. His research interests include intelligent systems, sensor data fusion, process data monitoring and diagnostics, and applied chaos theory and logistics.

Natarajan Gautam is an associate professor in the Industrial and Systems Engineering Department as well as in the Electrical Engineering Department at Texas A&M University. He received his Ph.D. and M.S. degrees in Operations Research from the University of North Carolina at Chapel Hill in 1997 and 1995 respectively. His research interests include optimal design, control and performance evaluation of stochastic systems, with special emphasis on service engineering, using techniques in queueing theory, applied probability, and optimization.