10-13-2006

Real-Time Divisible Load Scheduling for Cluster Computing

Xuan Lin
University of Nebraska, lxuan@cse.unl.edu

Ying Lu
University of Nebraska - Lincoln, ylu4@unl.edu

Jitender S. Deogun
University of Nebraska - Lincoln, jdeogun1@unl.edu

Steve Goddard
University of Nebraska – Lincoln, goddard@cse.unl.edu

University of Nebraska–Lincoln, Computer Science and Engineering
Technical Report TR-UNL-CSE-2006-0016
Issued Oct. 13, 2006

Technical Report UNL-CSE-2006-0016
Real-Time Divisible Load Scheduling for Cluster Computing

Xuan Lin, Ying Lu, Jitender Deogun, Steve Goddard
Department of Computer Science and Engineering
University of Nebraska - Lincoln
Lincoln, NE 68588
{*lxuan, ylu, deogun, goddard*}*@cse.unl.edu*

Abstract

Cluster computing has emerged as a new paradigm for solving large-scale problems. To enhance QoS and provide performance guarantees in cluster computing environments, various real-time scheduling algorithms and workload models have been investigated. Computational loads that can be arbitrarily divided into independent pieces represent many real-world applications. Divisible load theory (DLT) provides insight into distribution strategies for such computations. However, the problem of providing performance guarantees to divisible load applications has not yet been systematically studied. This paper investigates such algorithms for a cluster environment. Design parameters that affect the performance of these algorithms and scenarios when the choice of these parameters have significant effects are studied. A novel algorithmic approach integrating DLT and EDF (earliest deadline first) scheduling is proposed. For comparison, we also propose a heuristic algorithm. Intensive experimental results show that the application of DLT to real-time cluster-based scheduling leads to significantly better scheduling approaches.

1 Introduction

The dawn of the information age has changed how we solve important problems. Emerging computation and data intensive applications cannot be solved by a single stand-alone machine. This has led to the emergence of *cluster computing*—which harnesses the power of hundreds and thousands of machines—as a new paradigm for computing. However, as the size of a cluster increases, so does the complexity of resource management and maintenance. Automated performance control and resource management is crucial to achieve continued evolution of cluster computing. Current cluster scheduling practice is similar in sophistication to early supercomputer batch scheduling algorithms, and no consideration is given to desired quality-of-service (QoS) attributes. To fully avail the power of computational clusters, new scheduling theory that provides high performance, QoS assurance, and streamlined management of the cluster resources needs to be developed.

The challenge, however, in developing real-time scheduling theory for cluster computing is to support various types of cluster applications. Broadly speaking, computational loads submitted to a cluster are structured in two primary ways: *indivisible* and *divisible*. An indivisible load is essentially a sequential job and thus must be assigned to a single processor. The divisible loads are comprised of tasks that can be executed in parallel and can be further divided into two categories: *modularly divisible* and *arbitrarily divisible* loads. *Modularly divisible* loads are divided a priori into a certain number of subtasks and are often described by a task (or processing) graph. *Arbitrarily divisible* loads can be partitioned into an arbitrarily large number of load fractions. Examples of arbitrarily divisible loads can be easily found in high energy and particle physics. For example, the CMS (Compact Muon Solenoid) [10] and ATLAS (AToroidal LHC Apparatus) [5] projects, which are associated with the Large Hadron Collider (LHC) at CERN (European Laboratory for Particle Physics), execute cluster-based applications with arbitrarily divisible loads. Usually all elements in such computational loads demand an identical type of processing, and relative to the huge total computation, the processing on each individual element is infinitesimally small.

The problem of providing QoS or real-time guarantees for sequential and modularly divisible jobs in distributed systems has been studied extensively. Similarly, significant progress has been made in *divisible load theory (DLT)*[29]. However, despite the increasing importance of arbitrarily divisible applications [24], to the best of our knowledge, the real-time scheduling of arbitrarily divisible loads has not been systematically investigated.

Scheduling of arbitrarily divisible loads represents a problem of great significance for cluster-based research computing facilities such as the U.S. CMS Tier-2 sites [26]. (The CMS project will not be fully operational until 2007.) One of the management goals at the University of Nebraska-Lincoln (UNL) Research Computing Facility (RCF) is to provide a multi-tiered QoS scheduling framework in which applications "*pay*" according to the response time requested for each job [26]. Existing real-time cluster scheduling algorithms assume the existence of a task graph for all applications, which are not appropriate for arbitrarily divisible loads. To better manage these high-end clusters and control their performance, we need new real-time scheduling algorithms for arbitrarily divisible applications.

Four contributions are made in this paper. First, DLT is extended to compute the minimum number of processors required to meet an application's deadline. Second, based on this, a novel algorithmic approach integrating DLT and EDF (earliest deadline first) scheduling is proposed. For comparison, we also propose a heuristic algorithm. Third, important design parameters are identified that affect the performance of real-time divisible-load scheduling algorithms. Fourth, we systematically investigated the effects of these design parameters on a set of real-time scheduling algorithms, and show that the application of DLT to real-time, cluster-based scheduling leads to significantly better scheduling approaches.

The remainder of this paper is organized as follows. Related work is presented in Section 2. Section 3 describes both task and system models. In Section 4, real-time scheduling algorithms investigated in this paper are discussed. We evaluate the performance of algorithms in Section 5 and conclude the paper in Section 6.

2 Related Work

Development of commodity-based clusters and Grid computing has recently gained considerable momentum. By linking a large number of computers together, a cluster provides a cost-effective facility for solving complex problems. In a large-scale Grid, the resource management system (RMS), which provides real-time guarantees or QoS, is central to its operation.

Research has been carried out in utility-driven cluster computing [30, 25] to improve the value of utility delivered to users. Proposed cluster RMSs [8, 3] have addressed the scheduling of both sequential and parallel loads. The goal of those schemes is similar to ours: to harness the power of resources based on user objectives.

The scheduling models investigated for distributed or multiprocessor systems most often (e.g., [23, 22, 14, 1, 20, 13]) assume periodic or aperiodic sequential jobs that must be allocated to a single resource and executed by their deadlines. With the evolution of cluster computing, researchers have begun to investigate real-time scheduling of parallel applications on a cluster [32, 21, 12, 2, 4]. However, most of these studies assume the existence of some form of task graph to describe communication and precedence relations between computational units called subtasks (i.e., nodes in the task graph).

The most closely related work to our problem is scheduling algorithms for "scalable real-time tasks" running in a multiprocessor system presented in [16]. In that paper, like divisible loads, it is assumed that a task can be executed on more than one processor and as more processors are allocated to it, its pure computation time decreases monotonically. The paper notes that the decision on the number of processors allocated to tasks is an important factor in the design of parallel scheduling algorithms. However, the simulations described in the paper are limited and are favorably biased towards their proposed schemes. Therefore, their conclusions on comparing their proposed MWF (Maximum Workload derivative First) schemes with the EDF and FIXED algorithms [19, 6] hold true only in certain scenarios.

Our work differs significantly from previous work in real-time as well as cluster computing in both the task model assumed and in the comprehensiveness of our study. In this paper unlike the previous study [16], we do not assume task execution times are known a priori. Instead, we apply DLT to guide task partitioning, to derive its execution time function, and to compute the minimum number of processors required to meet its deadline.

DLT provides an in-depth study of distribution strategies for arbitrarily divisible loads [7, 24, 29]. The goal of DLT is to exploit parallelism in computational data so that the workload can be partitioned and assigned to several processors such that execution completes in the shortest possible time [7]. DLT has been previously applied to and implemented in Grid computing [31, 15, 28]. Complimentary to that work, our paper applies DLT in the design of real-time scheduling algorithms for cluster computing; specifically, DLT is applied in the partitioning of applications, such as CMS [10] and ATLAS [5], that execute on a large cluster.

3 Task and System Models

In this section we describe our task and system models briefly, and state assumptions related to these models.

Task Model. We assume a real-time aperiodic task model in which each aperiodic task T_i consists of a single invocation specified by the tuple (A_i, σ_i, D_i), where $A_i \geq 0$ is the arrival time of the task, $\sigma_i > 0$ is the total data size of the task, and $D_i > 0$ is its relative deadline. The absolute deadline of the task is given by $A_i + D_i$. Section 4.2 presents, in detail, how task execution time is dynamically computed based on total data size σ_i, resources allocated (i.e., processing nodes and bandwidth) and the partitioning method applied to parallelize the computation.

System Model. A cluster consists of a head node, denoted by P_0, connected via a switch to N processing nodes, denoted by P_1, P_2, ..., P_N. We assume that all processing nodes have the same computational power and all links from the switch to the processing nodes have the same bandwidth. The system model assumes a typical cluster environment in which the head node does not participate in computation. The role of the head node is to accept or reject incoming tasks, execute the scheduling algorithm, divide the workload and distribute data chunks to processing nodes. Since different nodes process different data chunks, the head node sequentially sends every data chunk to its corresponding processing node via the switch. We assume that data transmission does not happen in parallel, although it is straightforward to generalize our model and include the case where some pipelining of communication may occur. For the divisible loads we assume that tasks and subtasks are independent. Therefore, there is no need for processing nodes to communicate with each other.

According to divisible load theory, linear models are used to represent processing and transmission times [29]. In the simplest scenario, the computation time of a load σ is calculated by a cost function $Cp(\sigma) = \sigma C_{ps}$, where C_{ps} represents the time to compute a unit of workload on a single processing node. The transmission time of a load σ is calculated by a cost function $Cm(\sigma) = \sigma C_{ms}$, where C_{ms} is the time to transmit a unit of workload from the head node to a processing node. For many applications the output data is just a short message and is negligible, particularly considering the very large size of the input data. Therefore, in this paper we only model transfer of application input data but not the transfer of output data. The extension to consider the output data transfer using DLT is straightforward.

4 Algorithms

This section presents real-time scheduling algorithms for divisible loads. To develop the algorithms, we need to make three important decisions. The first is to adopt a scheduling policy to determine the order of execution for tasks (Section 4.1). The second decision is to determine the number n of processing nodes to allocate to each task and the third is to choose a strategy to partition the task among the allocated n nodes (Section 4.2).

4.1 Scheduling Policies

Three scheduling policies to determine the execution order of tasks are investigated: FIFO, EDF and MWF (Maximum Workload derivative First) [16]. The FIFO scheduling algorithm executes tasks following their order of arrival. EDF, a well-known real-time scheduling algorithm, orders tasks by their absolute deadlines. MWF is a real-time scheduling algorithm for divisible tasks.

The main rules of MWF are: 1) a task with the highest workload derivative (DC_i) is scheduled first; and 2) the number of nodes allocated to a task is kept as small as possible (n^{min}) without violating its deadline. Node assignment is discussed in Section 4.2. Here, we review how MWF determines task execution order and define the workload derivative metric, DC_i, where $W_i(n)$ is used to represent the workload (cost) of a task T_i when n processing nodes are assigned to it.

$$DC_i = W_i(n_i^{min} + 1) - W_i(n_i^{min}) \tag{4.1}$$

That is, $W_i(n) = n \times \mathcal{E}(\sigma_i, n)$, where $\mathcal{E}(\sigma_i, n)$ denotes the task's execution time (see Section 4.2 for $\mathcal{E}$'s calculation). Therefore, DC_i is the derivative of the task workload $W_i(n)$ at n_i^{min} (the minimum number of nodes needed by T_i to meet its deadline).

4.2 Node Assignment and Task Partitioning

We study two primary strategies for node assignment. First, *assign a task all N nodes* and thus try to finish the current task as early as possible. Second, *assign a task the minimum number n^{min} of nodes it needs to meet its deadline*, and thereby save resources for new tasks.

Similarly, two different partitioning methods are investigated: *Optimal Partitioning Rule* (OPR) (analyzed in Section 4.2.1), and *Equal Partitioning Rule* (EPR) (analyzed in Section 4.2.2). OPR is based on divisible load theory (DLT), which states that the optimal execution time is obtained when all nodes allocated to the task complete their computation at the same time [29]. For comparison, we propose EPR, based on a common practice of dividing a task into n equal-sized subtasks when the task is to be processed by n nodes.

The following notations, partially adopted from [29], are used in the analysis.

- $T = (A, \sigma, D)$: A divisible task, where A is the arrival time, σ is the data size, and D is the relative deadline.
- $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$: Data distribution vector, where n is the number of processing nodes allocated to the task, α_j is the data fraction allocated to the j^{th} node, i.e., $\alpha_j\sigma$, is the amount of data that is to be transmitted to the j^{th} node for processing, $0 < \alpha_j \leq 1$ and $\Sigma_{j=1}^{n}\alpha_j = 1$.
- C_{ms}: Cost of transmitting a unit workload.
- C_{ps}: Cost of processing a unit workload.

Based on our system model (Section 3) we have the following cost functions: the data transmission time on the j^{th} link is $C_m(\alpha_j\sigma) = \alpha_j\sigma C_{ms}$ and the data processing time on the j^{th} node is $C_p(\alpha_j\sigma) = \alpha_j\sigma C_{ps}$. We assume that the setup costs for initializing data transmission and processing are negligible. We have also investigated the case where the setup costs may be significant (see Appendix for details).

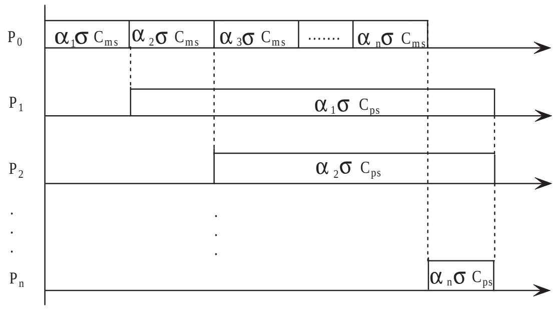


Figure 1: Time Diagram for OPR-Based Partitioning.

4.2.1 Optimal Partitioning Rule (OPR)

For a given task, we first analyze its execution time function, $\mathcal{E}(\sigma, n)$, assuming n nodes are to be allocated to process a total data size of σ. Then, we use the function to derive the minimum number, n^{min}, of nodes needed to meet the task's deadline.

Task Execution Time Analysis. Figure 1 shows an example task execution time diagram following OPR when n nodes are allocated to process the task. Let $\mathcal{E}$ denote *Task Execution Time*, which is a function of σ and n. We have,

$$\begin{aligned}
\mathcal{E}(\sigma,n) &= \alpha_1\sigma C_{ms} + \alpha_1\sigma C_{ps} && (4.2)\\
&= (\alpha_1+\alpha_2)\sigma C_{ms} + \alpha_2\sigma C_{ps} && (4.3)\\
&= (\alpha_1+\alpha_2+\alpha_3)\sigma C_{ms} + \alpha_3\sigma C_{ps} && (4.4)\\
&\dots\\
&= (\alpha_1+\alpha_2+\alpha_3+...+\alpha_n)\sigma C_{ms} + \\
&\quad \alpha_n\sigma C_{ps}. && (4.5)
\end{aligned}$$

From (4.2) and (4.3), we get

$$\begin{aligned}
\alpha_1 &= \alpha_2\frac{\sigma C_{ms}+\sigma C_{ps}}{\sigma C_{ps}} = \frac{\alpha_2}{\beta}, \quad \text{where}\\
\beta &= \frac{\sigma C_{ps}}{\sigma C_{ms}+\sigma C_{ps}} = \frac{C_{ps}}{C_{ms}+C_{ps}} && (4.6)
\end{aligned}$$

Note that $0 < \beta < 1$. It follows that $\alpha_2 = \beta\alpha_1$. Similarly, from (4.3) and (4.4), we have $\alpha_3 = \beta\alpha_2$, and therefore, $\alpha_3 = \beta^2\alpha_1$. This leads to a general formula: $\alpha_j = \beta^{j-1}\alpha_1$. Since α_j is the data fraction distributed to the j^{th} processing node, we have $\sum_{j=1}^{n}\alpha_j = 1$. Substituting α_j with $\beta^{j-1}\alpha_1$ in this equation, we obtain

$$\alpha_1 + \beta\alpha_1 + \beta^2\alpha_1 + ... + \beta^{n-1}\alpha_1 = 1.$$

Solving this equation, we get $\alpha_1 = \frac{1-\beta}{1-\beta^n}$. Substituting it into (4.2), we have

$$\mathcal{E}(\sigma,n) = \frac{1-\beta}{1-\beta^n}\sigma(C_{ms}+C_{ps}). \tag{4.7}$$

Derivation of n^{min}. Given $\mathcal{E}(\sigma,n)$, we can calculate the minimum number n^{min} of nodes required to meet a task's deadline.

Let $C(n)$ denote the task completion time function. Assuming that the task $T = (A, \sigma, D)$ has a start time s, then its completion time is $C(n) = s + \mathcal{E}(\sigma,n)$, which leads to

$$C(n) = s + \frac{1-\beta}{1-\beta^n}\sigma(C_{ms}+C_{ps}). \tag{4.8}$$

To meet a task's deadline means its completion time should satisfy the constraint that $C(n) \le A + D$. It follows that

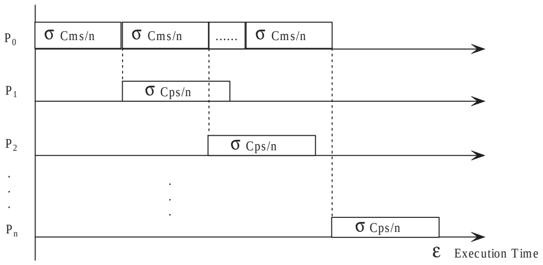


Figure 2: Time Diagram for EPR-Based Partitioning.

$$\begin{aligned} s+\frac{1-\beta}{1-\beta^n}\sigma(C_{ms}+C_{ps}) &\leq A+D, \quad \text{that is} \\ \frac{1-\beta}{1-\beta^n}\sigma(C_{ms}+C_{ps}) &\leq A+D-s. \end{aligned} \tag{4.9}$$

Since $1-\beta^n > 0$. Multiplying both sides of (4.9) by $(1-\beta^n)$, we get

$$(1-\beta)\sigma(C_{ms}+C_{ps}) \leq (1-\beta^n)(A+D-s). \tag{4.10}$$

If $A+D-s \leq 0$, the task will miss its deadline no matter how many nodes we assign to it and how we partition it. Such a task will be rejected because it fails the schedulability test of our scheduling algorithms (for details see Sec. 4.3). Thus, $A+D-s > 0$, and dividing both sides of (4.10) by $(A+D-s)$ we have

$$\begin{aligned} (1-\beta^n) &\geq \frac{(1-\beta)\sigma(C_{ms}+C_{ps})}{A+D-s}, \quad \text{thus} \\ \beta^n &\leq 1-\frac{(1-\beta)\sigma(C_{ms}+C_{ps})}{A+D-s} \\ &= 1-\frac{(1-\frac{C_{ps}}{C_{ms}+C_{ps}})\sigma(C_{ms}+C_{ps})}{A+D-s} \\ &= 1-\frac{(\frac{C_{ms}}{C_{ms}+C_{ps}})\sigma(C_{ms}+C_{ps})}{A+D-s} \\ &= 1-\frac{\sigma C_{ms}}{A+D-s} \end{aligned}$$

Let $\gamma = 1-\frac{\sigma C_{ms}}{A+D-s}$. Thus, $\beta^n \leq \gamma$. If $\gamma \leq 0$, starting task T at time s will not leave enough time even for its data transmission and therefore the task will be rejected as well. Thus, $\gamma > 0$, and it follows that $n \geq \frac{\ln\gamma}{\ln\beta}$. Since n, the number of nodes assigned, should be an integer, we have $n \geq \lceil\frac{\ln\gamma}{\ln\beta}\rceil$. Therefore, the minimum number of processing nodes that the task needs at time s to meet its deadline is $n^{min} = \lceil\frac{\ln\gamma}{\ln\beta}\rceil$ where γ is defined above and β in (4.6).

4.2.2 Equal Partitioning Rule (EPR)

To understand the merits of divisible load theory (DLT) in practical real-time cluster-based scheduling, we analyze EPR as a comparison. Similar to the analysis for DLT-based OPR, we derive the task execution time function and n^{min} for EPR.

Task Execution Time Analysis. Assuming n nodes are allocated to a task, an example task execution time diagram following the EPR is shown in Figure 2. By analyzing the diagram, we have $\mathcal{E}(\sigma,n) = \sigma C_{ms} + \alpha_n\sigma C_{ps}$, where $\alpha_n = \frac{1}{n}$. Thus,

$$\mathcal{E}(\sigma,n) = \sigma C_{ms} + \frac{\sigma C_{ps}}{n}. \tag{4.11}$$

Derivation of n^{min}. Assuming that the task $T = (A,\sigma,D)$ has a start time s, then the task completion time is $C(n) = s + \mathcal{E}(\sigma,n)$, which must satisfy the constraint that $C(n) \leq A+D$. That is,

$$s + \sigma C_{ms} + \frac{\sigma C_{ps}}{n} \leq A + D. \tag{4.12}$$

Thus,

$$\frac{\sigma C_{ps}}{A + D - s - \sigma C_{ms}} \leq n. \tag{4.13}$$

Therefore, following EPR, the minimum number of processing nodes that the task needs at time s to complete before its deadline is $n^{min} = \lceil \frac{\sigma C_{ps}}{A+D-s-\sigma C_{ms}} \rceil$.

4.3 Algorithm Framework

As is typical for dynamic real-time scheduling algorithms [11, 18, 22], when a task arrives, the scheduler dynamically determines if it is feasible to schedule the new task without compromising the guarantees for previously admitted tasks. The pseudocode for a general *Schedulability Test* is shown in Figure 3. It could be configured to generate various real-time divisible load scheduling algorithms by giving the design decisions on: 1) scheduling policy (FIFO, EDF or MWF), 2) node assignment method (assigning a task all N or its n^{min} nodes), and 3) task partitioning rule (OPR or EPR). Upon completion of the test, if all tasks are schedulable a feasible schedule is developed and the new task is accepted, otherwise, it is rejected[1].

By following the aforementioned framework, we generate ten algorithms: EDF-OPR-MN, EDF-OPR-AN, EDF-EPR-MN, EDF-EPR-AN, FIFO-OPR-MN, FIFO-OPR-AN, FIFO-EPR-MN, FIFO-EPR-AN, MWF-OPR-MN, and MWF-EPR-MN. The nomenclature of the algorithms, include three parts corresponding to the three algorithm design decisions. The first part denotes the scheduling policy adopted: EDF, FIFO or MWF. The second part represents the choice of the partitioning rule: DLT-based OPR or heuristic EPR. In the third portion of the name, MN means the algorithm assigns a task the minimum number of nodes needed to meet its deadline, and AN means the algorithm assigns all nodes. Since MWF always allocates a task n^{min} nodes, the algorithm only has the MN versions.

5 Performance Evaluation

The previous section proposed various real-time cluster-based scheduling algorithms for divisible loads. This section evaluates their performance relative to each other and to changes of various configuration parameters.

Cluster Configuration. We use a discrete simulator to simulate a range of clusters that are compliant to the system model presented in Section 3. For every simulation, three parameters, N, C_{ms} and C_{ps} are specified for a cluster.

Workload Generation. To generate task $T_i = (A_i, \sigma_i, D_i)$, we assume that the interarrival times follow an exponential distribution with a specified mean of $1/\lambda$, and task data sizes σ_i are assumed to be normally distributed with a specified mean of $Avg\sigma$ and a standard deviation equal to the mean. Task relative deadlines are assumed to be uniformly distributed in the range of $[\frac{AvgD}{2}, \frac{3AvgD}{2}]$, where $AvgD$ is the mean relative deadline. To specify $AvgD$, a new term $DCRatio$ is introduced. It is defined as the ratio of mean deadline to mean minimum execution time (cost), that is $\frac{AvgD}{\mathcal{E}(Avg\sigma,N)}$, where $\mathcal{E}(Avg\sigma, N)$ is the task execution time computed with Eq (4.7) assuming the task has an average data size $Avg\sigma$ and runs on all N processing nodes. Given $DCRatio$, the cluster size N and the average data size $Avg\sigma$, $AvgD$ is implicitly specified as $DCRatio \times \mathcal{E}(Avg\sigma, N)$. In this way, by $DCRatio$, task relative deadlines are specified relating to the average task execution time. In addition, a task relative deadline D_i is chosen to be larger than its minimum execution time $\mathcal{E}(\sigma_i, N)$. In summary, we specify the following parameters for every simulation: $(N, C_{ms}, C_{ps}, 1/\lambda, Avg\sigma, DCRatio)$.

To analyze how loaded a cluster is for a simulation, we define another metric $SystemLoad$. It is derived from the specified parameters as

$$SystemLoad = \frac{\mathcal{E}(Avg\sigma, N)}{\lambda},$$

which corresponds to

$$SystemLoad = \frac{TotalTaskNumber \times \mathcal{E}(N, Avg\sigma)}{TotalSimulationTime}.$$

Sometimes, we specify $SystemLoad$ for the simulation instead of average interarrival time $1/\lambda$. Configuring $(N, C_{ms}, C_{ps}, SystemLoad,$ is equivalent to specifying $(N, C_{ms}, C_{ps}, 1/\lambda, Avg\sigma, DCRatio)$, because

$$1/\lambda = \frac{SystemLoad}{\mathcal{E}(Avg\sigma, N)}.$$

[1] Rejection in the cluster environment means that the system administrator (or a program proxy) will negotiate with the client for a feasible task deadline, and the job will be rescheduled with modified parameters.

Data Structure:

- $n_i^{min}(t)$ — the minimum number of processing nodes needed to finish T_i before its deadline, assuming it is dispatched at time t.

- AvailableNodesList <t_k, AN_k> — a list of number of available nodes along with the time, where t_k is the time and AN_k is the number of available nodes.

Pseudocode:

```
boolean Schedulability-Test(T )
    TempTasksList ← T + AdmittedTasksQueue

    // According to the chosen scheduling policy:
    // EDF, FIFO or MWF (Decision #1)
    order TempTasksList

    // Obtain the available nodes information
    generate AvailableNodesList < $t_k$,$AN_k$ >

    //Initialization
    ScheduledTaskList ←$\phi$

    while TempTaskList != $\phi$
        remove $T_i(A_i, \sigma_i, D_i)$ from TempTasksList

        // According to the chosen node assignment policy:
        // assigning a task $n^{min}$ or N nodes (Decision #2)
        identify the earliest time $t_k$ when the
        available nodes $AN_k \geq n_i^{min}(t_k)$ or $AN_k \geq N$

        // Set scheduled starting time
        $s_i \leftarrow t_k$

        // According to the chosen partitioning rule:
        // OPR or EPR (Decision #3), calculate $\varepsilon$ following
        // Eq (4.7) in Section 4.2.1 or Eq (4.11) in Section 4.2.2,
        // and set the expected completion time
        $e_i \leftarrow \varepsilon(\sigma_i, n_i) + s_i$

        if $e_i > A_i + D_i$
            return false        // Deadline misses


        put $T_i(A_i, \sigma_i, D_i, s_i, n_i, e_i)$ into ScheduledTaskList

        update AvailableNodesList
    end while

    /* All tasks in the cluster are schedulable */
    AdmittedTasksQueue← ScheduledTaskList

    return true

end Schedulability Test()
```

Figure 3: Schedulability Test.

To evaluate the performance of the real-time scheduling algorithms, we use the metric, *Task Reject Ratio*, which is the ratio of the number of tasks rejected by a real-time scheduling algorithm to the total number of tasks arriving at the cluster. The smaller the *Task Reject Ratio*, the better the real-time scheduling algorithm.

For all figures in this paper, a point on a curve corresponds to the average performance value of ten simulations. In the ten runs, the same parameters $(N, C_{ms}, C_{ps}, 1/\lambda, Avg\sigma, DCRatio)$ are specified but different random numbers are generated for task arrival times A_i, data sizes σ_i, and deadlines D_i. For each simulation, the $TotalSimulationTime$ is 10,000,000 time units, which is sufficiently long.

In Section 4, we identified three important decisions on *Task Partitioning*, *Node Assignment*, and *Scheduling Policy* in designing real-time, cluster-based scheduling algorithms for divisible loads. The next three subsections evaluate and compare the algorithms proposed in Section 4 and respectively investigate the scenarios where each of these three decisions matters.

5.1 OPR vs. EPR Partitioning

We first evaluate the performance of the following real-time scheduling algorithms with respect to the two proposed partioning rules (OPR and EPR): EDF-OPR-MN v.s. EDF-EPR-MN, EDF-OPR-AN v.s. EDF-EPR-AN, FIFO-OPR-MN v.s. FIFO-EPR-MN, FIFO-OPR-AN v.s. FIFO-EPR-AN, and MWF-OPR-MN v.s. MWF-EPR-MN. Only the comparisons of EDF-OPR-MN v.s. EDF-EPR-MN and EDF-OPR-AN v.s. EDF-EPR-AN are presented here. The performance results for the other pairs are similar (see Figure 13-20 in Appendix for details).

5.1.1 Simulation Modeling

For our basic simulation model we chose the following parameters: number of processing nodes in the cluster, $N = 16$; unit data transmission time, $C_{ms} = 1$; unit data processing time, $C_{ps} = 100$; $SystemLoad$ changes in the range $\{0.1, 0.2, \cdots, 1.0\}$; Average data size, $Avg\sigma = 200$; and the ratio of the average deadline to the average execution time, $DCRatio = 2$. Our simulation has a three-fold objective. *First*, we want to verify our hypothesis that it is advantageous to apply DLT in real-time cluster-based scheduling. *Second*, we study the effects of $DCRatio$, and *third*, we want to investigate effects of the processing speed.

5.1.2 Merits of DLT for Cluster Scheduling

To study the merits of DLT we employ our basic simulation model without any changes. The four curves in Figure 4a show the *Task Reject Ratio* of the four algorithms: EDF-OPR-MN, EDF-EPR-MN, EDF-OPR-AN, and EDF-EPR-AN. Observe that EDF-OPR-MN always leads to a lower *Task Reject Ratio* than EDF-EPR-MN. Similarly, observe that EDF-OPR-AN always achieves a lower *Task Reject Ratio* than EDF-EPR-AN. These simulation results confirm our hypothesis that it is advantageous to apply DLT in real-time, cluster-based scheduling algorithms. DLT provides an optimal task partitioning, which leads to minimum task execution times, and as a result the cluster can satisfy a larger number of task deadlines.

We carried out the same type of simulations by changing, one at a time, the following cluster or workload parameters: cluster size N, unit transmission time C_{ms}, and average data size $Avg\sigma$. Results are similar to Figure 4a, where algorithms with OPR partitioning always perform better than algorithms with EPR partitioning (see Figure 10, 6d - 6d, 11, and 12 in Appendix for details).

5.1.3 Effects of $DCRatio$

To study the effects of the $DCRatio$, we use the same configuration as the basic simulation except that we vary the $DCRatio$ over the range $\{2, 3, 10, 20, 100\}$. For sake of readability, Figure $4b$ only shows the performance of EDF-OPR-AN and EDF-EPR-AN with DCRatio = 2, 10, and 100 (see Figure 6, 7, and 8 in Appendix for other results). Corresponding to different combinations of algorithm and DCRatio, six curves are produced. Again, Figure $4b$ shows that the algorithm with OPR partitioning performs better. In addition, we can see as $DCRatio$ increases, the performance of EDF-EPR-AN becomes closer to that of EDF-OPR-AN. This is because the higher the $DCRatio$, the looser the task relative deadlines are. Consequently, the worse execution times caused by a non-optimal partition, like EPR, will have less impact on the algorithms' performance. In particular, when $DCRatio$ is extremely high (100), the two algorithms perform almost the same.

5.1.4 Effects of Processing Speed

To study effects of the processing speed, we vary C_{ps} over the range $\{10, 50, 100, 500, 1000, 5000, 10000\ \}$. The larger the C_{ps}, the slower the computation. Figure $4c$ shows the results of EDF-OPR-MN and EDF-EPR-MN with $C_{ps} = 10$ and 10000 respectively (see Figure 9 in Appendix for other results).

From the figure, we can see that the algorithm with OPR partitioning (EDF-OPR-MN) still outperforms the algorithm with EPR partitioning (EDF-EPR-MN). However, as the processing speed decreases, i.e., C_{ps} increases, the differences between the two algorithms becomes less and less significant. In particular, when the computation is extremely slow ($C_{ps} = 10000$), we can see that the curves for the two algorithms are almost overlapped, indicating non-differentiable *Task Reject Ratios*. To demonstrate this point, let us assume C_{ps} is so large that the ratio of C_{ms} to C_{ps} is approaching 0. In the analysis of OPR, we showed that β from Eq (4.6) will approach 1 in this case, causing the data fractions allocated to processing nodes, $\alpha_1, \alpha_2, \cdots, \alpha_n$, to all be close to $\frac{1}{n}$. Therefore, OPR and EPR will perform the same in this case.

From the aforementioned intensive experiments, we conclude that no matter what the system parameters are, the algorithms with DLT-based partitioning (OPR) always perform better than the ones with the equal-sized partitioning heuristic (EPR). This shows that it is beneficial to apply divisible load theory in real-time, cluster-based scheduling.

5.2 N v.s. n^{min} Nodes

In this subsection, we compare and analyze the real-time scheduling algorithms with different node assignment methods. We investigate the performance difference in algorithms assigning all N nodes to every task (ALG-AN) v.s. those assigning the minimum number n^{min} of nodes needed to meet a task's deadline (ALG-MN).

The relative performance of EDF-OPR-MN v.s. EDF-OPR-AN is shown in Section 5.2.1. It is noteworthy that in contrast to the results in [16] comparing MWF(-MN) and FIXED(-AN) algorithms, our initial data seem to indicate that EDF-OPR-AN outperforms EDF-OPR-MN most of the time. To gain insight into the performance results, in Section 5.2.2 we carry out rigorous analysis of a simplified scenario where a scheduling algorithm always assigns K nodes ($K < N$) to a periodic divisible task. This analysis sheds new light on possible scenarios where algorithms assigning n^{min} nodes (ALG-MN) perform better than those assigning all N nodes (ALG-AN).

5.2.1 Initial Comparison

In Section 4.2, we have explained the rational behind the two node-assignment strategies: an ALG-AN tries to finish the current task as soon as possible, while an ALG-MN saves resources for new tasks.

For the N node assignment strategy, the problem is it causes higher parallel execution overhead than the n^{min} node assignment counterpart. For the cluster model investigated (Section 3), the higher the transmission cost (C_{ms}), the greater the overhead. As shown in Figure 1, the node idle time due to data transmission is one type of parallel execution overhead.

On the other hand, the n^{min} node assignment strategy, trying to save resources for new tasks, will not improve performance if there is no task coming in the near future. In that case, some number of nodes (potentially $N - n^{min}$ of them) are left idle and their processing cycles are wasted. Since the larger the interarrival time $1/\lambda$, the less frequent are the tasks arrivals, we believe that ALG-MN will loose its performance gain over ALG-AN as the interarrival time $1/\lambda$ increases.

We conducted intensive simulations, comparing EDF-OPR-MN v.s. EDF-OPR-AN, to verify the aforementioned analysis and present some conclusive simulation results.

As explained, EDF-OPR-AN leads to higher overhead than EDF-OPR-MN, and the larger C_{ms}, the higher the parallel overhead. Thus, we expect that as C_{ms} increases, the performance of EDF-OPR-AN should be affected more than that of EDF-OPR-MN. Figure 4d shows the relative performance of the two algorithms (*Task Reject Ratio* of EDF-OPR-MN $-$ *Task Reject Ratio* of EDF-OPR-AN) in a simulation where we gradually increase the value of C_{ms}. We can see as C_{ms} gets larger, the difference between the *Task Reject Ratios* of EDF-OPR-MN and EDF-OPR-AN decreases, indicating that the relative performance of EDF-OPR-MN v.s. EDF-OPR-AN improves. Interestingly, for this simulation, the curve is above 0 for most of the cases, indicating EDF-OPR-MN performs worse than EDF-OPR-AN.

The next group of simulations verifies our prediction that EDF-OPR-MN will perform worse relative to EDF-OPR-AN as task interarrival time $1/\lambda$ increases. As demonstrated in Figure 4e, with the increasing of the interarrival time, *Task Reject Ratio* of EDF-OPR-MN $-$ *Task Reject Ratio* of EDF-OPR-AN also increases. The relative *Task Reject Ratio* curve once again lies above 0, meaning EDF-OPR-MN performs worse than EDF-OPR-AN.

The results from our initial simulations contradict the conclusion drawn by [16] that the n^{min} node assignment strategy performs better than the maximum node assignment strategy.

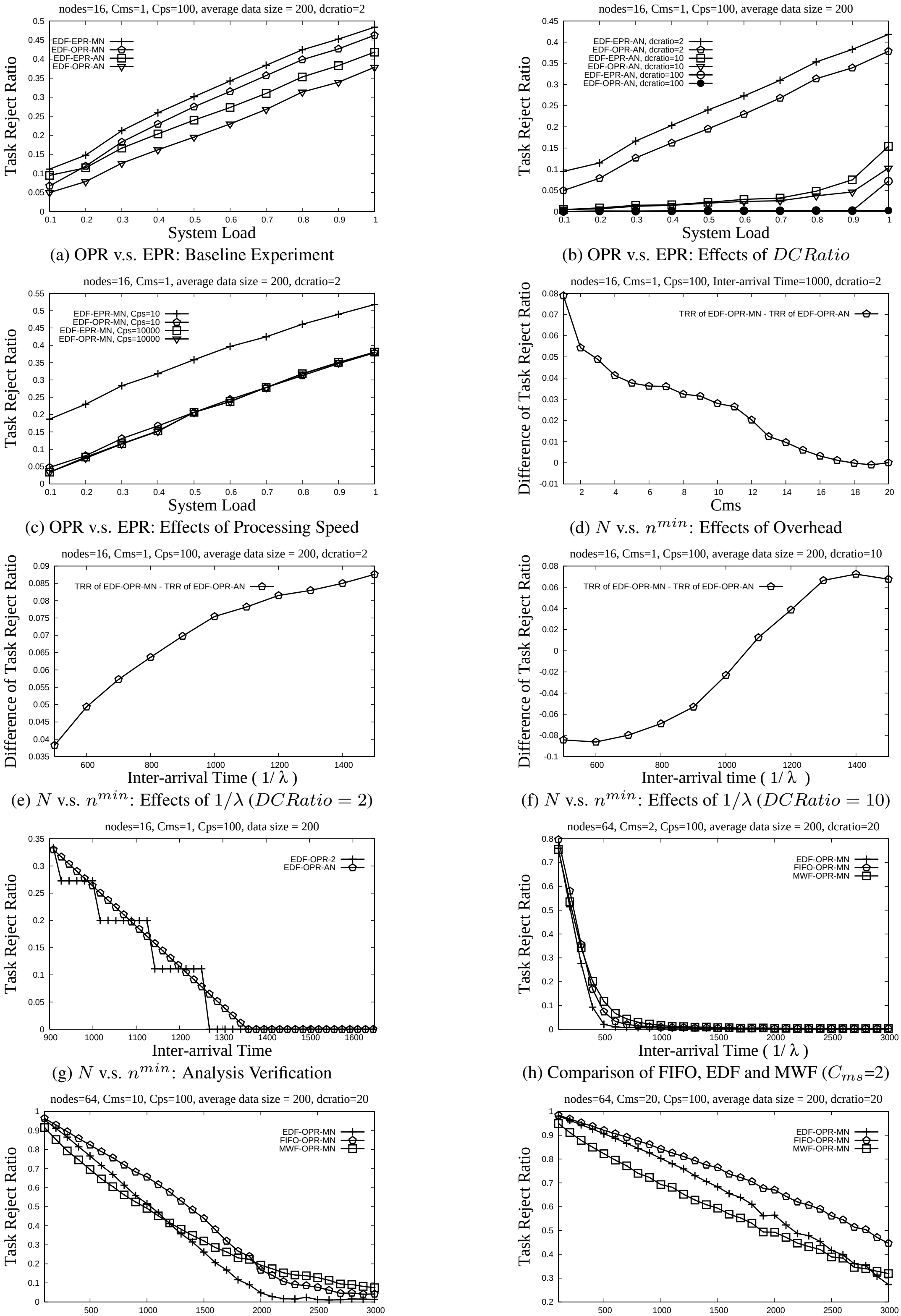
We believe that there should be some scenarios where ALG-MN performs better than ALG-AN, while in the other scenarios (for instance, Figures 4d and 4e) the reverse is true. To show that there are cases where ALG-MN outperforms ALG-AN, we purposely configure a simulation where ALG-MN should have improved performance.

For this simulation, we choose a similar configuration as that in Figure 4e except changing the $DCRatio$ from 2 to 10. By increasing the $DCRatio$, we have longer relative deadlines compared to the mean execution time. For an ALG-MN, a longer deadline leads to a smaller n^{min} of nodes allocated to a task, thus smaller overhead. While for an ALG-AN, its node assignment and resulting overhead will not be affected by deadlines, since a task is always assigned all N nodes. Therefore, we believe, as $DCRatio$ increases and ALG-MN's overhead decreases, ALG-MN might perform better than ALG-AN.

Figure 4f validates our analysis. We can successfully create some scenarios (those with interarrival time smaller than 1000 time units), by increasing $DCRatio$ from 2 to 10, where EDF-OPR-MN outperforms EDF-OPR-AN.

5.2.2 Analysis

In this subsection, we study a simplified scenario where there is only one periodic divisible task. A new algorithm, ALG-K that always assigns K nodes ($K < N$) to every task is investigated, which is compared with an ALG-AN. We demonstrate that with this simple analysis we could identify some scenarios where an ALG-MN will be better than the corresponding ALG-AN when handling aperiodic divisible tasks.



(a) OPR v.s. EPR: Baseline Experiment

(b) OPR v.s. EPR: Effects of $DCRatio$

(c) OPR v.s. EPR: Effects of Processing Speed

(d) N v.s. n^{min}: Effects of Overhead

(e) N v.s. n^{min}: Effects of $1/\lambda$ ($DCRatio = 2$)

(f) N v.s. n^{min}: Effects of $1/\lambda$ ($DCRatio = 10$)

(g) N v.s. n^{min}: Analysis Verification

(h) Comparison of FIFO, EDF and MWF (C_{ms}=2)

(i) Comparison of FIFO, EDF and MWF (C_{ms}=10)

(j) Comparison of FIFO, EDF and MWF (C_{ms}=20)

Figure 4: Performance Evaluation.

The system model is the same as described in Section 3, while the task model is simplified: there is only one periodic task T. In every period P we assume that a subtask (also called a job) of T with a fixed data size σ and a relative deadline D arrives at the cluster. For such system and task models, we have proved the following theorems.

Theorem 5.1 *Let $\mathcal{E}(\sigma, n)$ denote the execution time of a task with a data size σ running on n nodes. We have $\mathcal{E}(\sigma, 1) < N\mathcal{E}(\sigma, N)$.*

Proof According to $Divisible\ Load\ Theory$, the execution times are

$$\mathcal{E}(\sigma, 1) = \sigma(C_{ms} + C_{ps}) \tag{5.14}$$

$$\mathcal{E}(\sigma, N) = \frac{1-\beta}{1-\beta^N}\sigma(C_{ms} + C_{ps}) \tag{5.15}$$

where β is defined in Eq. (4.6). Dividing Eq.(5.14) by Eq.(5.15), we get

$$\begin{aligned} \frac{\mathcal{E}(\sigma,1)}{\mathcal{E}(\sigma,N)} &= \frac{1-\beta^N}{1-\beta} \\ &= 1 + \beta + \beta^2 + \cdots + \beta^{N-1} \\ &< N \end{aligned}$$

That is, $\mathcal{E}(\sigma, 1) < N\mathcal{E}(\sigma, N)$.

Corollary 5.2 *If first job arrives at time 0, A_i, the arrival time of the ith job, is $(i-1)P$. That is*

$$A_i = (i-1)P. \tag{5.16}$$

Assertion 5.3 *When algorithm ALG-AN is applied, if a task T has a period P such that $P \geq \mathcal{E}(\sigma, N)$, then the $waiting\ time$ of the i^{th} job is 0; On the other hand, if $P < \mathcal{E}(\sigma, N)$, then the $waiting\ time$ of the i^{th} job is $(i-1)(\mathcal{E}(\sigma, N) - P)$.*

According to the above Assertion, all deadlines will be met by algorithm ALG-AN if the task period $P \geq \mathcal{E}(\sigma, N)$.

Theorem 5.4 *When an ALG-AN is applied, if $P < \mathcal{E}(\sigma, N)$ and D is finite, some subtasks are doomed to miss their deadlines.*

Proof This follows directly from Assertion 5.3: when i is large enough, ALG-AN will miss some job's deadline sooner or later.

Next, we will prove that under certain conditions, when ALG-K is applied, no subtasks will miss their deadlines.

Corollary 5.5 *In any time period $[t_x, t_x + L)$, for a task T with a period P there are at most $\lceil \frac{L}{P} \rceil$ job arrivals.*

Lemma 5.6 *When algorithm ALG-K is applied, if $P \geq \frac{K\mathcal{E}(\sigma,K)}{N-K}$ and $C_{ps} > (N-1)C_{ms}$, all subtasks of the periodic task will start as soon as they arrive.*

Proof We prove this Lemma by contradiction. As explained, ALG-K always assigns K nodes ($K < N$) to every job. Assume J_x is the first job that can not start as soon as it arrives. Then at the arrival time A_x of job J_x, the number of available nodes must be less than K, indicating there are another $\lfloor \frac{N}{K} \rfloor$ number of jobs running. We order these jobs by their arrival time and get a job list $J_1, J_2, \cdots$. Assuming the arrival time of the first job J_1 is A_1, then in the time period $[A_1, A_x]$ there are $\lfloor \frac{N}{K} \rfloor + 1$ number of job arrivals. Since job J_1 is still running at time A_x, we have $A_x < A_1 + \mathcal{E}(\sigma, K)$. Therefore, in the time period $[A_1, A_1 + \mathcal{E}(\sigma, K))$ there are at least $\lfloor \frac{N}{K} \rfloor + 1$ number of job arrivals. On the other hand, since $P \geq \frac{K\mathcal{E}(\sigma,K)}{N-K}$, according to Corollary 5.5, the number m of job arrivals in $[A_1, A_1 + \mathcal{E}(\sigma, K))$ should be at most $\lfloor \frac{N}{K} \rfloor$. This is shown as follows

$$\begin{aligned} m &\leq \lceil \frac{\mathcal{E}(\sigma,K)}{\frac{K\mathcal{E}(\sigma,K)}{N-K}} \rceil \\ &= \lceil \frac{N-K}{K} \rceil \\ &= \lceil \frac{N}{K} \rceil - 1 \\ &\leq \lfloor \frac{N}{K} \rfloor \end{aligned}$$

We find a contradiction.

K	Range	EDF-OPR-K TRR	EDF-OPR-AN TRR
1	[1263,1359)	0	0.0184
2	[1269,1359)	0	0.0263
4	[1282,1359)	0	0.0251
8	[1307,1359)	0	0.0187

Table 1: For a $(N = 16, C_{ms} = 1, C_{ps} = 100)$ cluster, the verification of the derived range where EDF-OPR-K outperforms EDF-OPR-AN.

K	Range
1	[316, 425)
2	[318, 425)
3	[335, 425)
4	[321, 425)
5	[350, 425)
6	[357, 425)
7	[366, 425)
8	[327, 425)

Table 2: For a $(N = 64, C_{ms} = 1, C_{ps} = 100)$ cluster, the derived range where EDF-OPR-K outperforms EDF-OPR-AN.

Theorem 5.7 *When algorithm ALG-K is applied, if $P \geq \frac{K\mathcal{E}(\sigma,K)}{N-K}$, $C_{ps} > (N-1)C_{ms}$, and $D \geq \mathcal{E}(\sigma, K)$, then all subtasks of the periodic task will meet their deadlines.*

Proof According to Lemma5.6, all subtasks will start as soon as they arrive. When deadline $D \geq \mathcal{E}(\sigma, K)$, all subtasks will meet their deadlines.

From Theorem 5.4 and 5.7, we conclude that if $P \in [\frac{k\mathcal{E}(\sigma,K)}{N-k}, \mathcal{E}(\sigma, N))$, $C_{ps} > (N-1)C_{ms}$ and $D \geq \mathcal{E}(\sigma, K)$ for some $K \in [1, N]$, an ALG-K will perform better than the corresponding ALG-AN.

We carry out simulations to verify this conclusion. A periodic task T that includes subtasks with size $\sigma = 200$ and deadline $D = \mathcal{E}(\sigma, 2)$ is simulated to run in a $(N = 16, C_{ms} = 1, C_{ps} = 100)$ cluster by EDF-OPR-2 or EDF-OPR-AN. According to the above conclusion, if the task period $P \in [1263, 1359)$, EDF-OPR-2 should perform better than EDF-OPR-AN. Simulations are carried out with the task period changing from 900 to 1700. Figure 4g shows the results, which demonstrate that EDF-OPR-2 has 0 *Task Reject Ratio* when the task period P falls in $[1263, 1359)$ range, outperforming EDF-OPR-AN. The derived result is verified.

To demonstrate that the above conclusion is also true for aperiodic tasks, we relax the fixed-period assumption. For all the simulated scenarios, the task interarrival times could be different but are always kept in the derived range where EDF-OPR-K is guaranteed to perform better than EDF-OPR-AN. Table 1 presents the simulation results. As expected, under these controlled scenarios, EDF-OPR-K algorithms perform better than EDF-OPR-AN.

Leveraging the Analysis Results. In the last simulation of this subsection, we demonstrate how the insight gained from the above analysis could be leveraged to derive the scenarios where ALG-MN is guaranteed to outperform its ALG-AN counterpart. For a $(N = 64, C_{ms} = 1, C_{ps} = 100)$ cluster, we derive the ranges where EDF-OPR-K, $K \in (1, 2, \cdots, 8)$ outperform EDF-OPR-AN (Table 2). We can see that the common subrange of the 8 ranges is $[366, 425)$. Thus, if task interarrival times fall into that range and the task relative deadlines are long enough that ALG-MN always generate $n^{min} \leq 8$, then ALG-MN will perform better than ALG-AN.

We conduct two simulations, one comparing EDF-OPR-MN v.s. EDF-OPR-AN and the other comparing FIFO-OPR-MN v.s. FIFO-OPR-AN. A scenario as described above is created. Simulation results show that for such a configuration, the *Task Reject Ratios* of EDF-OPR-MN and FIFO-OPR-MN are both 0 while the *Task Reject Ratios* of EDF-OPR-AN and FIFO-OPR-AN are 0.0523 and 0.0564 respectively. We thus verify under the derived conditions an ALG-MN indeed performs better than its ALG-AN counterpart.

The models in this subsection all assume tasks have the same data size. In the future we plan to extend the analysis to a more general task model.

5.3 FIFO, EDF and MWF

In this subsection, we examine the effects of different execution order policies and compare algorithms FIFO-OPR-MN, EDF-OPR-MN v.s. MWF-OPR-MN.

Recall that the MWF (Maximum Workload derivative First) algorithm, proposed in [16], executes the task with highest workload derivative (DC_i) first, and thus reduces the total workload (cost) of all scheduled tasks. In [16] MWF is compared with EDF and shown that MWF performs better than EDF. Moreover, the authors claim that MWF is likely to be the best choice for on-line scheduling of divisible tasks.

We conducted intensive simulations and a systematic study of the three execution order strategies. Our data cast some doubts on the conclusion drawn in [16] that the MWF algorithm is the best choice. Our hypothesis is that MWF performs well when the parallel execution overhead (workload) of the tasks is significant compared to their pure computation. To test our theory, a group of simulations is designed to study how changing parallel overhead affects the performance of scheduling algorithms. In the 20 simulations, we gradually change the data transmission cost (C_{ms}) from 1 to 20, while keeping the data processing cost (C_{ps}) constant. Since the bigger the C_{ms}, the higher the parallel execution overhead, for the 20 simulations with C_{ms} changing from 1 to 20 the task overhead increases. According to our theory, MWF should perform better than EDF and FIFO when C_{ms} increases.

Figures 4h, 4i and 4j show the results for simulations where $C_{ms} = 2$, 10 and 20 respectively. As observed, when C_{ms} is small (Figure 4h), the *Task Reject Ratio* curve of EDF-OPR-MN lies below that of MWF-OPR-MN, indicating EDF execution order performs better. As C_{ms} increases (Figures 4i and 4j), the relative performance of the two algorithms changes. When C_{ms} increases to 20 (Figure 4j), MWF-OPR-MN outperforms EDF-OPR-MN. These data match our analysis and verify our hypothesis that MWF performs better than EDF and FIFO as workload parallel overhead increases.

Interestingly, for all 20 simulations, EDF-OPR-MN always performs better than FIFO-OPR-MN, while in some scenarios like the one in Figure 4h, MWF-OPR-MN performs even worse than FIFO-OPR-MN.

Another observation is that as the interarrival time increases, the performance of the three algorithms become non-differentiable (see Figure 4h). As the interarrival time gets larger and the system load smaller, either EDF and MWF generate the same execution order as FIFO, or the load is so light that the choice of execution order does not matter anymore.

In summary, our data indicate that the best choice of execution order policy depends on individual system and workload conditions. It might be appropriate and desirable to have an adaptive scheduling algorithm where policies can be configured with changing conditions. In the future, we plan to study real cluster workload traces and further investigate this problem.

6 Conclusion

In this paper, we address the problem of providing deterministic QoS to arbitrarily divisible applications executing in a cluster. Four contributions are made. *First*, we extend DLT to compute the minimum number of processors required to meet an application deadline. *Second*, based on this, a novel algorithmic approach integrating DLT and EDF scheduling is proposed. *Third*, important design parameters are identified that affect the performance of real-time divisible-load scheduling algorithms. *Finally*, we systematically investigated the effects of these design parameters on a set of real-time scheduling algorithms, and show that the application of DLT to real-time, cluster-based scheduling leads to significantly better scheduling approaches.

References

[1] T. F. Abdelzaher and V. Sharma. A synthetic utilization bound for aperiodic tasks with resource requirements. In *Proc. of 15th Euromicro Conference on Real-Time Systems*, pages 141–150, Porto, Portugal, July 2003.

[2] A. Amin, R. Ammar, and A. E. Dessouly. Scheduling real time parallel structure on cluster computing with possible processor failures. In *Proc of 9th IEEE International Symposium on Computers and Communications*, pages 62–67, July 2004.

[3] Y. Amir, B. Awerbuch, A. Barak, R. S. Borgstrom, and A. Keren. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Trans. on Parallel and Distributed Systems*, 11(7):760+, 2000.

[4] R. A. Ammar and A. Alhamdan. Scheduling real time parallel structure on cluster computing. In *Proc. of 7th IEEE International Symposium on Computers and Communications*, pages 69–74, Taormina, Italy, July 2002.

[5] ATLAS (AToroidal LHC Apparatus) Experiment, CERN (European Lab for Particle Physics). Atlas web page. http://atlas.ch/.

[6] D. Babbar and P. Krueger. On-line hard real-time scheduling of parallel tasks on partitionable multiprocessors. In *ICPP*, pages 29–38, 1994.

[7] V. Bharadwaj, T. G. Robertazzi, and D. Ghose. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[8] B. N. Chun and D. E. Culler. Market-based proportional resource sharing for clusters. Technical Report UCB/CSD-00-1092, EECS Department, University of California, Berkeley, 2000.

[9] G. Chun, H. Dail, H. Casanova, and A. Snavely. Benchmark probes for grid assessment. In *IPDPS*, 2004.

[10] Compact Muon Solenoid (CMS) Experiment for the Large Hadron Collider at CERN (European Lab for Particle Physics). Cms web page. http://cmsinfo.cern.ch/Welcome.html/.

[11] M. L. Dertouzos and A. K. Mok. Multiprocessor online scheduling of hard-real-time tasks. *IEEE Trans. Softw. Eng.*, 15(12):1497–1506, 1989.

[12] M. Eltayeb, A. Dogan, and F. Özgüner. A data scheduling algorithm for autonomous distributed real-time applications in grid computing. In *Proc. of 33rd International Conference on Parallel Processing*, pages 388–395, Montreal, Canada, Aug 2004.

[13] S. Funk and S. Baruah. Task assignment on uniform heterogeneous multiprocessors. In *Proc of 17th Euromicro Conference on Real-Time Systems*, pages 219–226, July 2005.

[14] D. Isovic and G. Fohler. Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. In *Proc. of 21st IEEE Real-Time Systems Symposium*, Orlando, FL, Nov 2000.

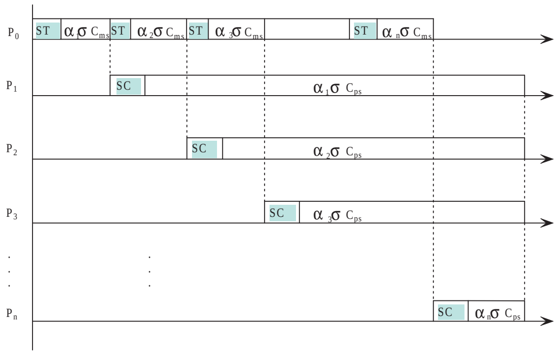


Figure 5: Timing Diagram of System with Setup Cost.

[15] S. Kim and J. B. Weissman. A genetic algorithm based approach for scheduling decomposable data grid applications. In *Proc. of International Conference on Parallel Processing*, pages 406–413, Montreal, Canada, Aug 2004.

[16] W. Y. Lee, S. J. Hong, and J. Kim. On-line scheduling of scalable real-time tasks on multiprocessor systems. *J. Parallel Distrib. Comput.*, 63(12):1315–1324, 2003.

[17] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for cluster computing. *Technical Report TR-UNL-CSE-2006-0016, University of Nebraska-Lincoln*, 2006.

[18] G. Manimaran and C. S. R. Murthy. An efficient dynamic scheduling algorithm for multiprocessor real-time systems. *IEEE Trans. on Parallel and Distributed Systems*, 9(3):312–319, 1998.

[19] P. Mohapatra. Dynamic real-time task scheduling on hypercubes. *J. of Parallel and Distributed Computing*, 46(1):91–100, 1997.

[20] P. Pop, P. Eles, Z. Peng, and V. Izosimov. Schedulability-driven partitioning and mapping for multi-cluster real-time systems. In *Proc. of 16th Euromicro Conference on Real-Time Systems*, pages 91–100, July 2004.

[21] X. Qin and H. Jiang. Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems. In *Proc. of 30th International Conference on Parallel Processing*, pages 113–122, Valencia, Spain, Sep 2001.

[22] K. Ramamritham, J. A. Stankovic, and P. fei Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Trans. on Parallel and Distributed Systems*, 1(2):184–194, Apr 1990.

[23] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Trans. Comput.*, 38(8):1110–1123, 1989.

[24] T. G. Robertazzi. Ten reasons to use divisible load theory. *Computer*, 36(5):63–68, 2003.

[25] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: a computational economy-based job scheduling system for clusters. *Software: Practice and Experience*, 34(6):573–590, 2004.

[26] D. Swanson. Personal communication. Director, UNL Research Computing Facility (RCF) and UNL CMS Tier-2 Site, Aug 2005.

[27] TERAGRID. http://www.teragrid.org/.

[28] K. van der Raadt, Y. Yang, and H. Casanova. Practical divisible load scheduling on grid platforms with apst-dv. In *Proc. of 19th International Parallel and Distributed Processing Symposium*, Denver, CA, Apr 2005.

[29] B. Veeravalli, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17, 2003.

[30] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience*, accepted in Sep 2005.

[31] D. Yu and T. G. Robertazzi. Divisible load scheduling for grid computing. In *Proc. of IASTED International Conference on Paralle and Distributed Computing and Systems*, Los Angeles, CA, Nov 2003.

[32] L. Zhang. Scheduling algorithm for real-time applications in grid environment. In *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, Oct 2002.

APPENDIX

Task Execution Time Analysis for OPR with Setup Costs. In Section 4.2.1, we have analyzed for optimal partitioning rule (OPR) the task execution time function and the minimum n^{min} number of nodes needed to meet a task's deadline. There, we assume that the setup costs for initializing data transmission and processing are negligible. In this appendix, we present the OPR analysis for the case where the setup costs are significant.

The setup cost of communication comes from physical network latencies, network protocol overhead, or middleware overhead. In the TeraGrid project [27], the network speed can be up to 40GBit/Sec with latency around 100ms, which means around 1/3 of the time required to send 1GB of data is due to latency. [9] shows that the setup cost for computation can be up to 25 seconds in practice, which is not negligible for small tasks.

To integrate the communication and computation setup costs in the analysis, we add the following two notations.

- ST: The setup time (cost) for the head node to initialize a communication on a link.
- SC: The setup time (cost) for a processing node to initialize a computation.

The goal of the analysis remains the same, to derive the task execution time function and the minimum number of nodes required to meet a task's deadline.

Task Execution Time Analysis. Considering the setup costs, the data processing time on the j^{th} node is modeled as $C_p(\alpha_j\sigma) = SC + \alpha_j\sigma C_{ps}$, and the data transmission time on the j^{th} link is $C_m(\alpha_j\sigma) = ST + \alpha_j\sigma C_{ms}$. Figure 5 shows an example task execution time diagram following OPR when n nodes are allocated to a task and setup costs are modeled. Analyzing the time diagram, we derive the *Task Execution Time* $\mathcal{E}$, a function of σ and n as follows

$$\begin{aligned}
\mathcal{E}(\sigma, N) &= (ST + \alpha_1\sigma C_{ms}) + (SC + \alpha_1\sigma C_{ps}) && (6.17)\\
&= 2ST + (\alpha_1 + \alpha_2)\sigma C_{ms} + \\
&\quad (SC + \alpha_2\sigma C_{ps}) && (6.18)\\
&= 3ST + (\alpha_1 + \alpha_2 + \alpha_3)\sigma C_{ms} + \\
&\quad (SC + \alpha_3\sigma C_{ps}) && (6.19)\\
&\dots \\
&= (n-1)ST + && (6.20)\\
&\quad (\alpha_1 + \alpha_2 + \alpha_3 + ... + \alpha_n)\sigma C_{ms} + \\
&\quad (SC + \alpha_n\sigma C_{ps})
\end{aligned}$$

From (6.17) and (6.18), we have $\alpha_2 = \alpha_1\beta - \phi$, where

$$\begin{aligned}
\beta &= \frac{C_{ps}}{C_{ms} + C_{ps}}, \quad \text{and} \\
\phi &= \frac{ST}{\sigma(C_{ms} + C_{ps})}. && (6.21)
\end{aligned}$$

Similarly, from (6.18) and (6.19), we get $\alpha_3 = \alpha_2\beta - \phi$, and therefore $\alpha_3 = \alpha_1\beta^2 - \beta\phi - \phi$, leading to the general formula

$$\begin{aligned}
\alpha_j &= \alpha_1\beta^{j-1} - \Sigma_{k=0}^{j-2}\beta^k\phi, \quad \text{Thus,} \\
\alpha_j &= \alpha_1\beta^{j-1} - \frac{1-\beta^{j-1}}{1-\beta}\phi.
\end{aligned}$$

Now, substituting α_j with $(\alpha_1\beta^{j-1} - \frac{1-\beta^{j-1}}{1-\beta}\phi)$ in equation $\sum_{j=1}^{n}\alpha_j = 1$, we get

$$\begin{aligned}
\Sigma_{j=1}^{n}(\alpha_1\beta^{j-1} - \frac{1-\beta^{j-1}}{1-\beta}\phi) &= 1 \\
\Longrightarrow \Sigma_{j=0}^{n-1}(\alpha_1\beta^{j} - \frac{1-\beta^{j}}{1-\beta}\phi) &= 1.
\end{aligned}$$

A solution to the above equation leads to

$$\alpha_1 = \frac{1-\beta}{1-\beta^n} + \frac{n\phi}{1-\beta^n} - \frac{\phi}{1-\beta}$$

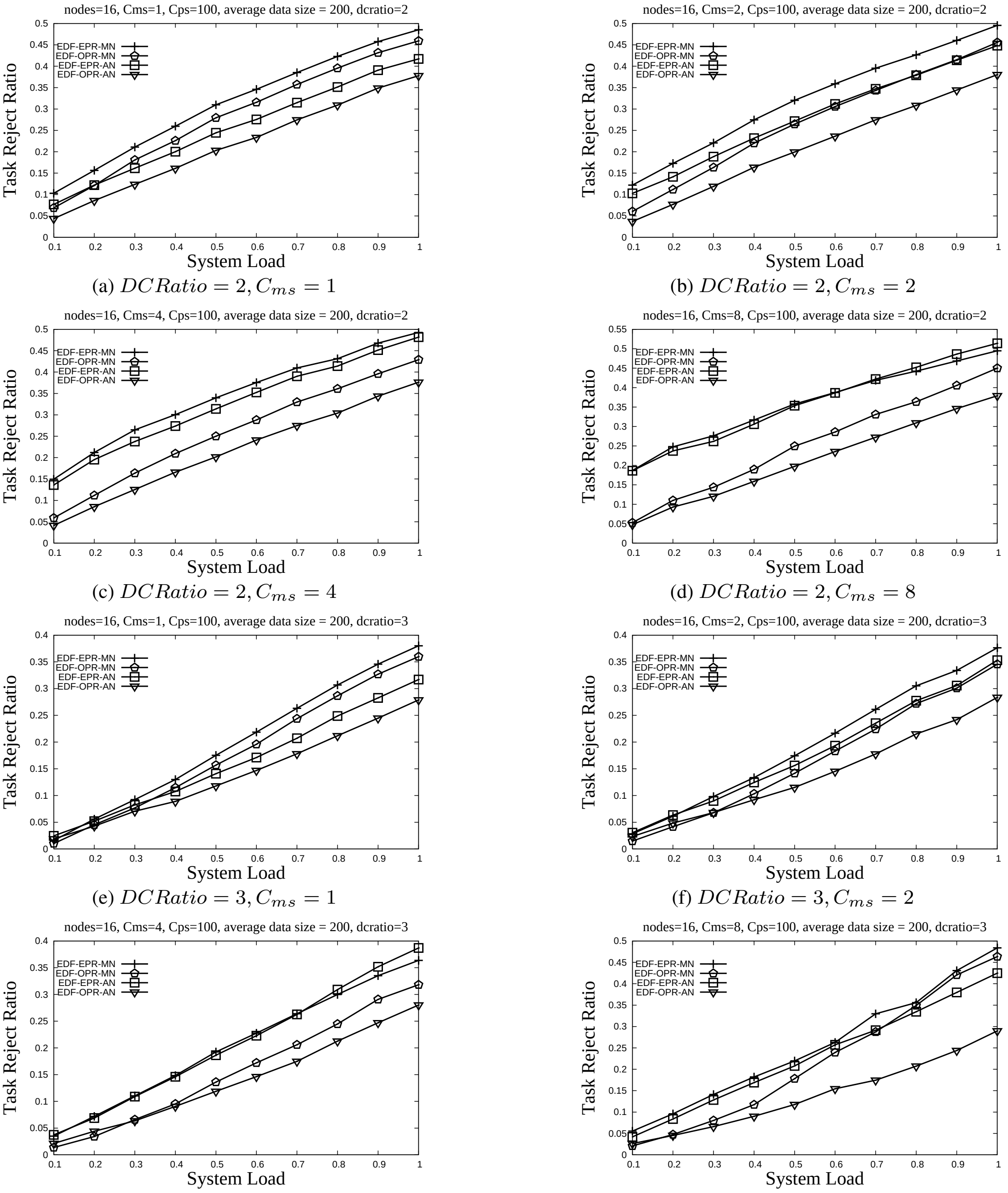
let

$$\mathcal{B}(n) = \frac{1-\beta}{1-\beta^n} + \frac{n\phi}{1-\beta^n} - \frac{\phi}{1-\beta}, \tag{6.22}$$

it follows that $\mathcal{E}(\sigma, N) = ST + SC + \sigma(C_{ms} + C_{ps})\mathcal{B}(n)$.

Derivation of n^{min}. If task $T = (A, \sigma, D)$ has a start time s, then to meet its deadline $\mathcal{E}(\sigma, N) \leq A + D - s$ should be satisfied. That is,

$$ST + SC + \sigma(C_{ms} + C_{ps})\mathcal{B}(n) \leq A + D - s. \tag{6.23}$$



(a) $DCRatio = 2, C_{ms} = 1$

(b) $DCRatio = 2, C_{ms} = 2$

(c) $DCRatio = 2, C_{ms} = 4$

(d) $DCRatio = 2, C_{ms} = 8$

(e) $DCRatio = 3, C_{ms} = 1$

(f) $DCRatio = 3, C_{ms} = 2$

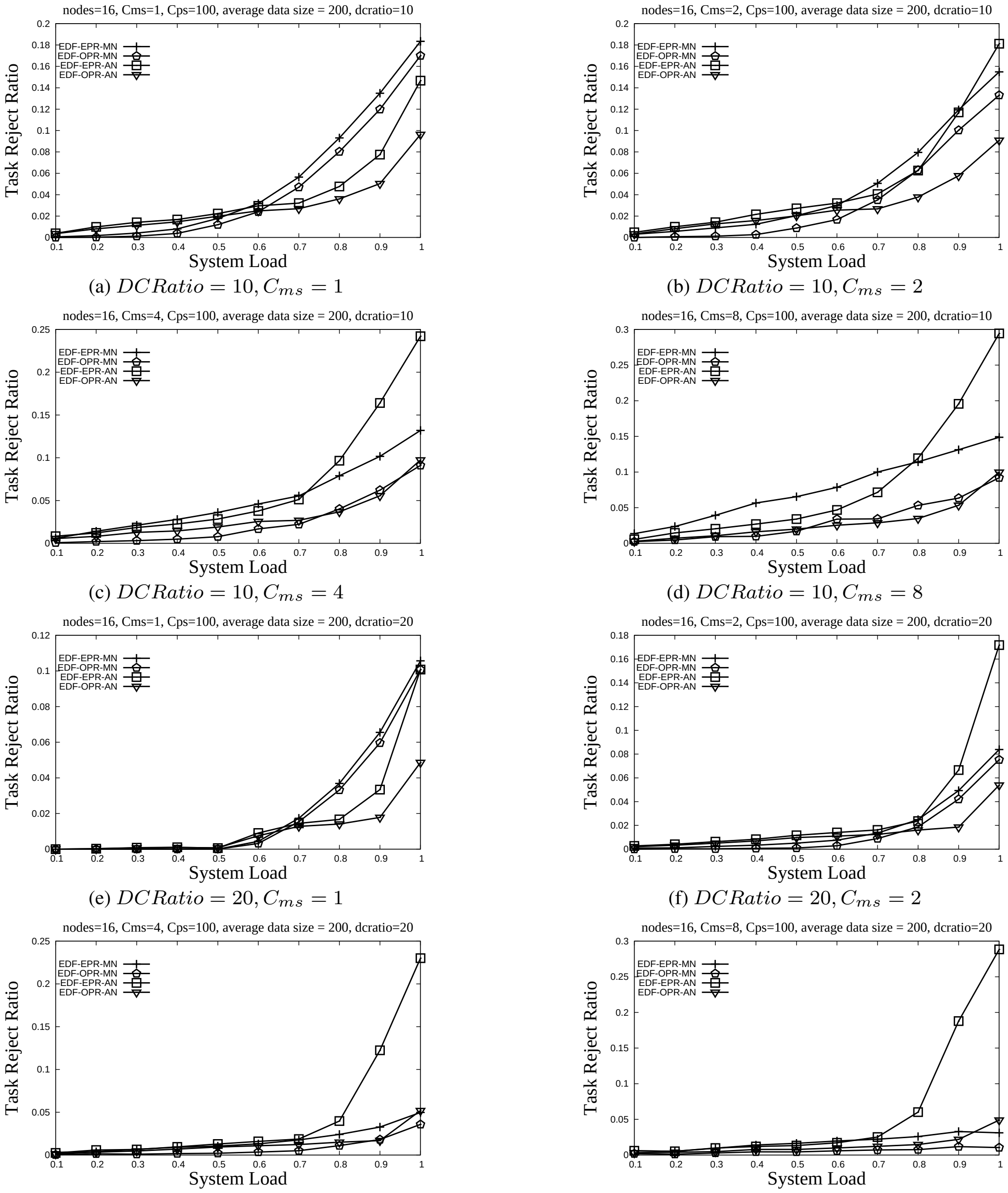
(g) $DCRatio = 3, C_{ms} = 4$

(h) $DCRatio = 3, C_{ms} = 8$

Figure 6: EDF-OPR v.s. EDF-EPR,$DCRatio = 2, 3$

This constraint can be solved numerically. The smallest integer n that satisfies the constraint is the minimum number n^{min} of nodes that needs to be assigned to task T at time s to satisfy its deadline.

Note that the model without setup costs (Section 4.2.1) is a special case of this model, where $ST = SC = 0$. Accordingly, $\phi = \frac{ST}{\sigma(C_{ms}+C_{ps})} = 0$ in that case. Thus, we reduce constraint (6.23) to $\sigma(C_{ms} + C_{ps})\frac{1-\beta}{1-\beta^n} \leq A + D - s$, the constraint (4.9) derived in Section 4.2.1.



(a) $DCRatio = 10, C_{ms} = 1$

(b) $DCRatio = 10, C_{ms} = 2$

(c) $DCRatio = 10, C_{ms} = 4$

(d) $DCRatio = 10, C_{ms} = 8$

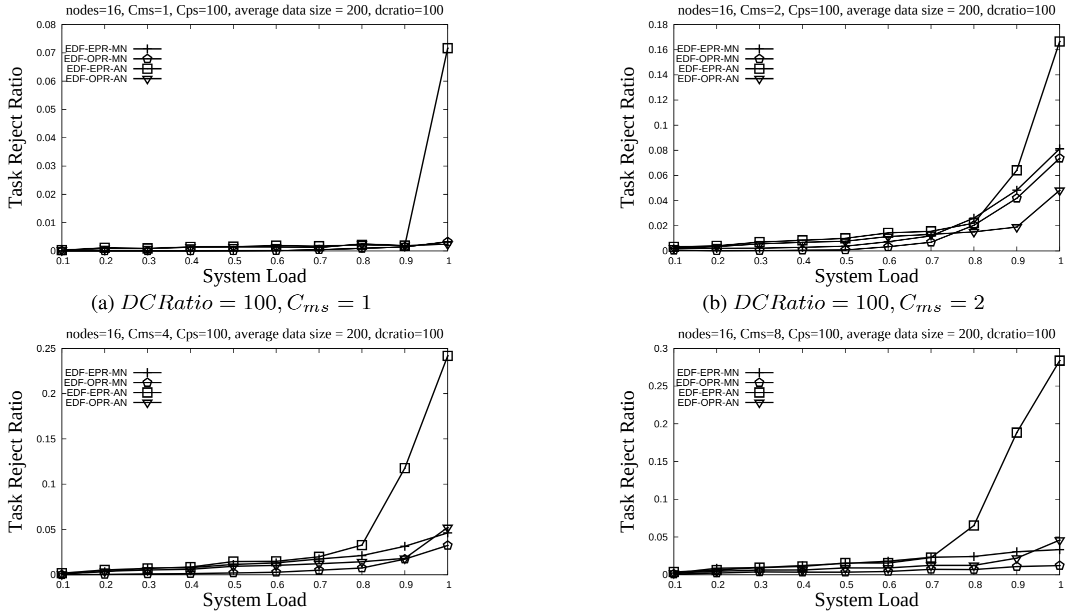
(e) $DCRatio = 20, C_{ms} = 1$

(f) $DCRatio = 20, C_{ms} = 2$

(g) $DCRatio = 20, C_{ms} = 4$

(h) $DCRatio = 20, C_{ms} = 8$

Figure 7: EDF-OPR v.s. EDF-EPR,$DCRatio = 10, 20$



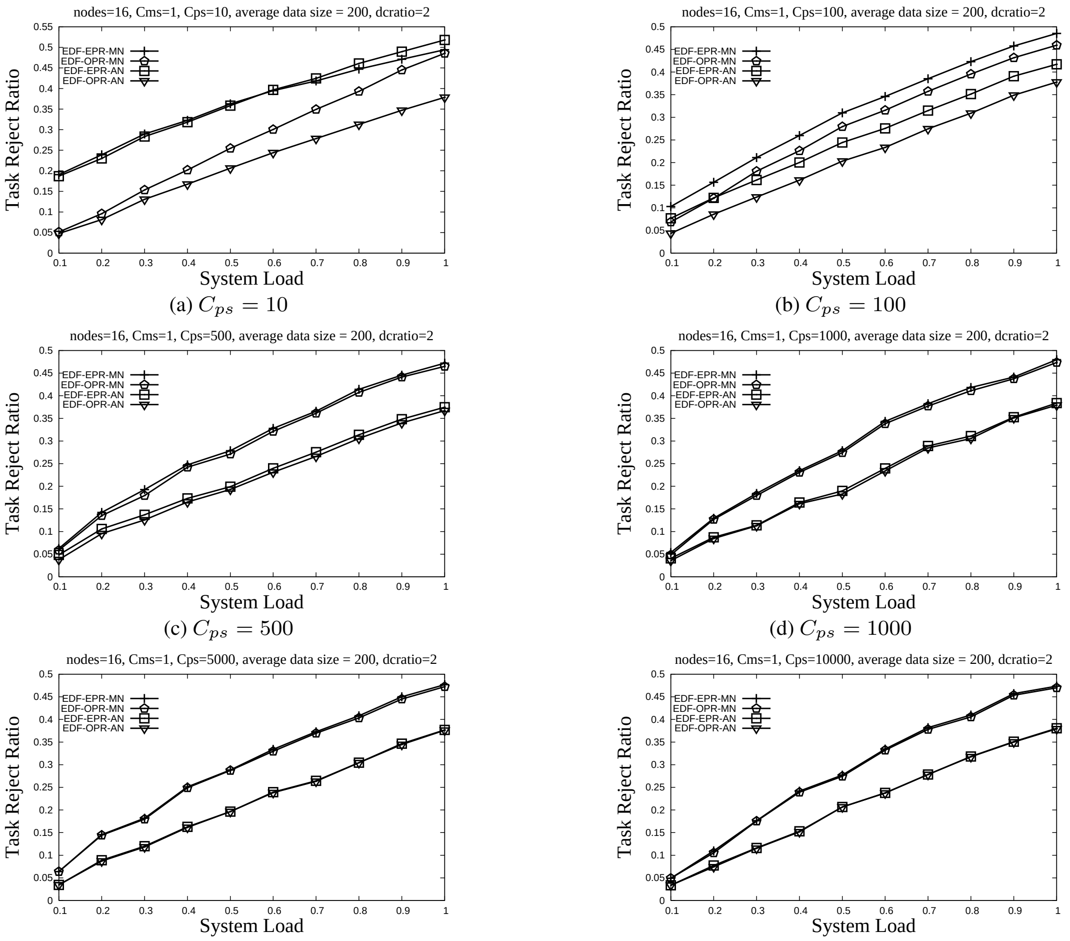
(a) $DCRatio = 100, C_{ms} = 1$

(b) $DCRatio = 100, C_{ms} = 2$

(c) $DCRatio = 100, C_{ms} = 4$

(d) $DCRatio = 100, C_{ms} = 8$

Figure 8: EDF-OPR v.s. EDF-EPR,$DCRatio = 100$



(a) $C_{ps} = 10$

(b) $C_{ps} = 100$

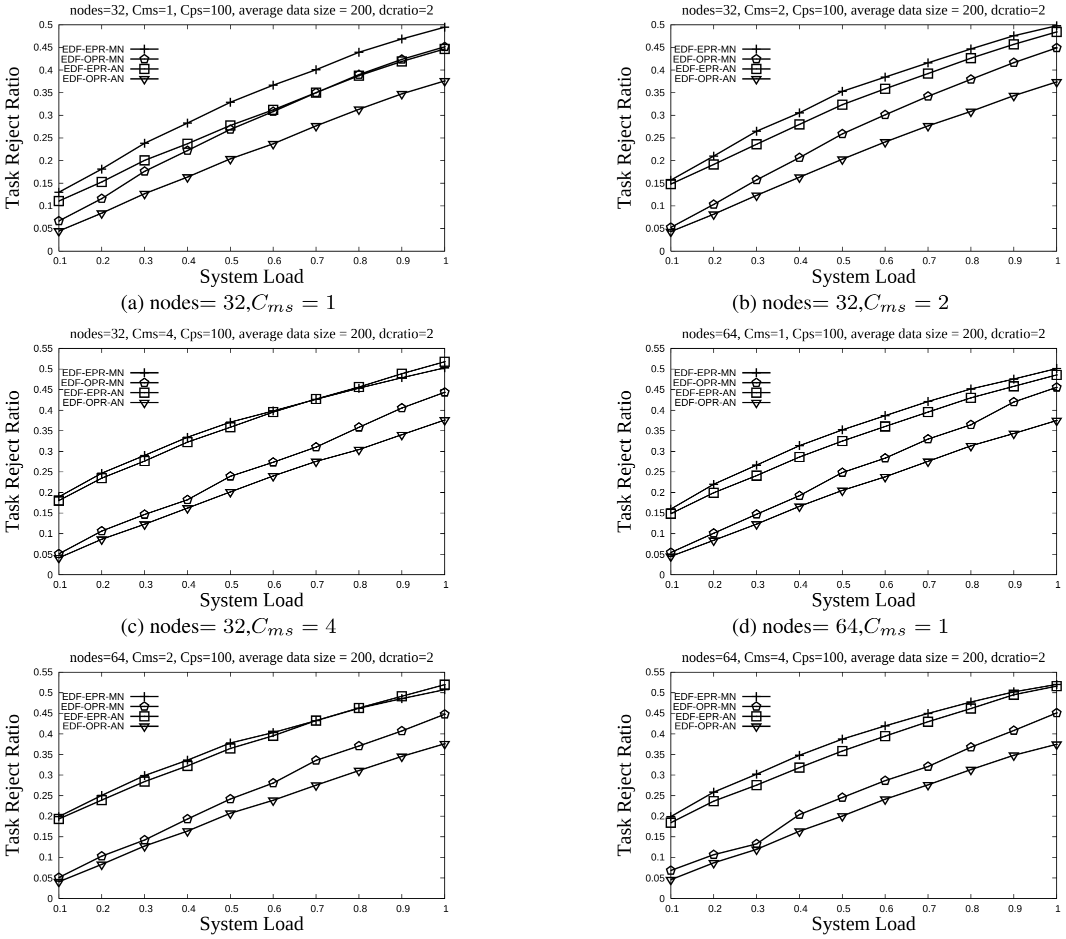
(c) $C_{ps} = 500$

(d) $C_{ps} = 1000$

(e) $C_{ps} = 5000$

(f) $C_{ps} = 10000$

Figure 9: EDF-OPR v.s. EDF-EPR,CPU



(a) nodes= 32,$C_{ms} = 1$

(b) nodes= 32,$C_{ms} = 2$

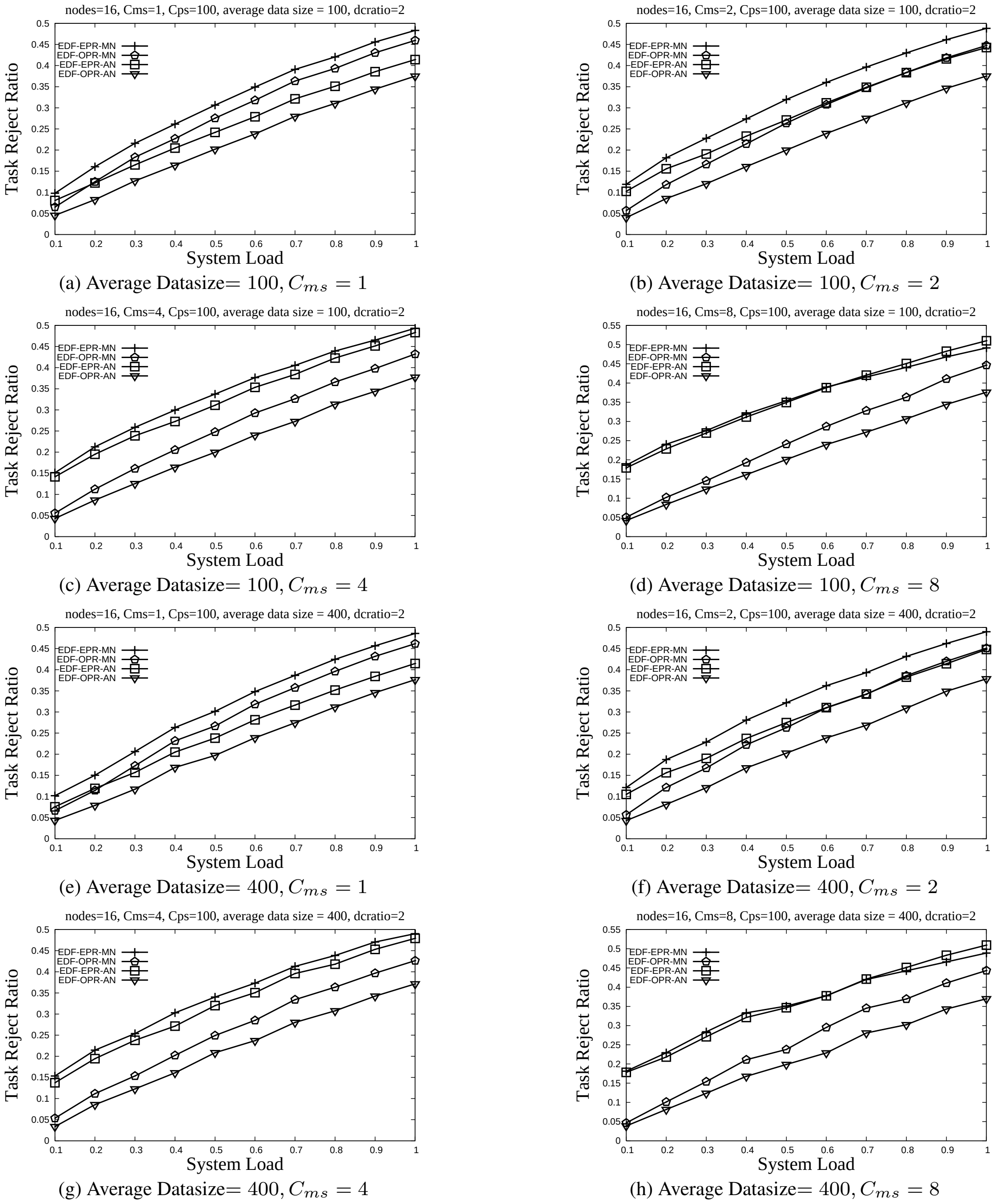
(c) nodes= 32,$C_{ms} = 4$

(d) nodes= 64,$C_{ms} = 1$

(e) nodes= 64,$C_{ms} = 2$

(f) nodes= 64,$C_{ms} = 4$

Figure 10: EDF-OPR v.s. EDF-EPR, Nodes=32,64



(a) Average Datasize$= 100, C_{ms} = 1$

(b) Average Datasize$= 100, C_{ms} = 2$

(c) Average Datasize$= 100, C_{ms} = 4$

(d) Average Datasize$= 100, C_{ms} = 8$

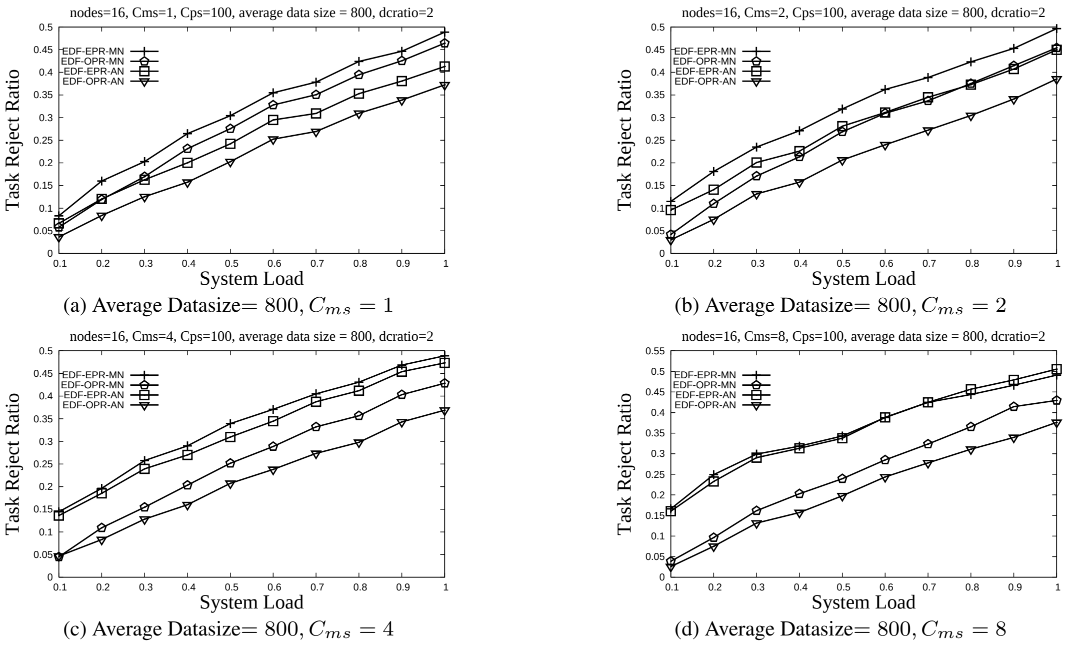
(e) Average Datasize$= 400, C_{ms} = 1$

(f) Average Datasize$= 400, C_{ms} = 2$

(g) Average Datasize$= 400, C_{ms} = 4$

(h) Average Datasize$= 400, C_{ms} = 8$

Figure 11: EDF-OPR v.s. EDF-EPR, Average Datasize = 100, 400



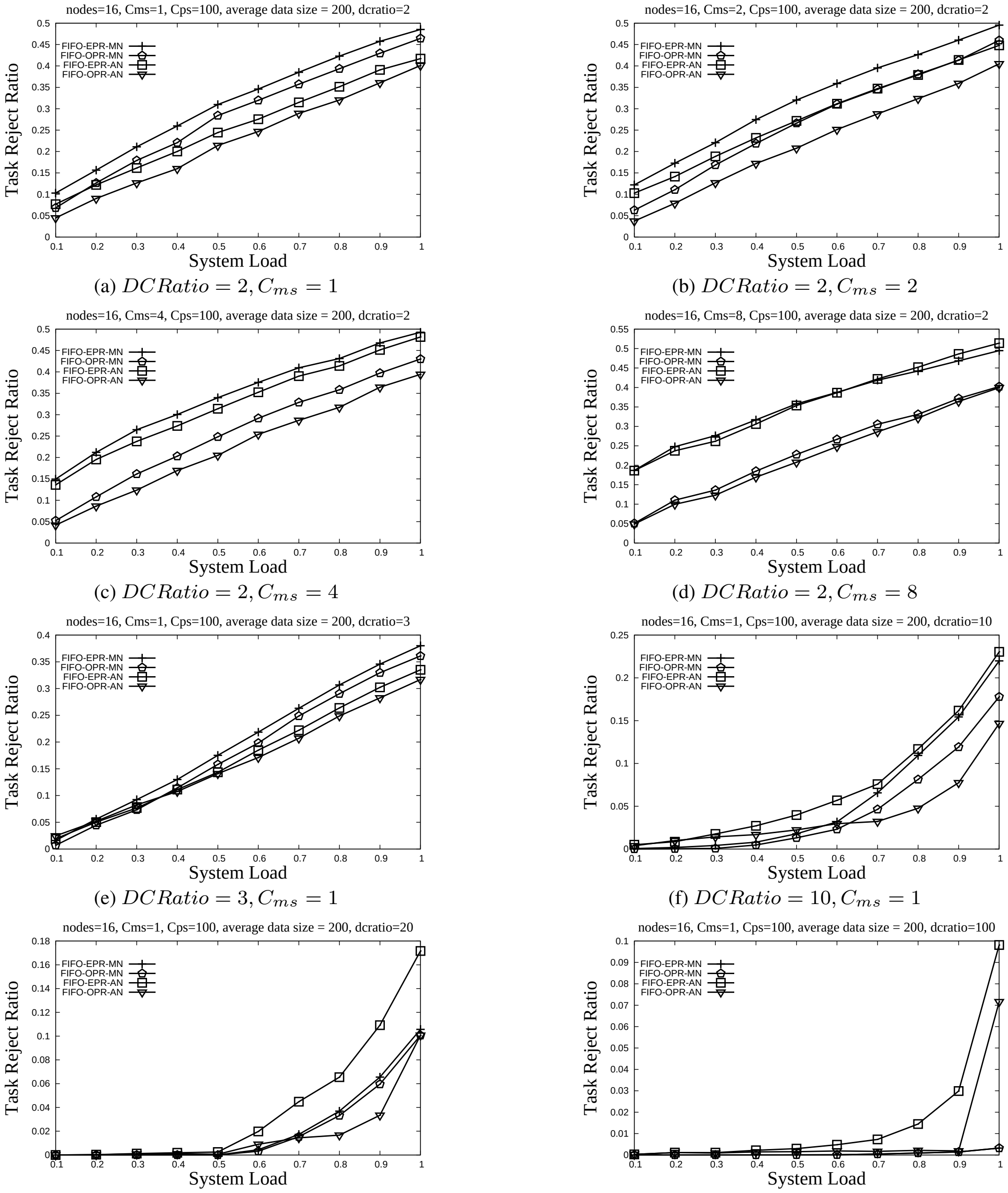
(a) Average Datasize$= 800, C_{ms} = 1$

(b) Average Datasize$= 800, C_{ms} = 2$

(c) Average Datasize$= 800, C_{ms} = 4$

(d) Average Datasize$= 800, C_{ms} = 8$

Figure 12: EDF-OPR v.s. EDF-EPR, Average Datasize = 800



(a) $DCRatio = 2, C_{ms} = 1$

(b) $DCRatio = 2, C_{ms} = 2$

(c) $DCRatio = 2, C_{ms} = 4$

(d) $DCRatio = 2, C_{ms} = 8$

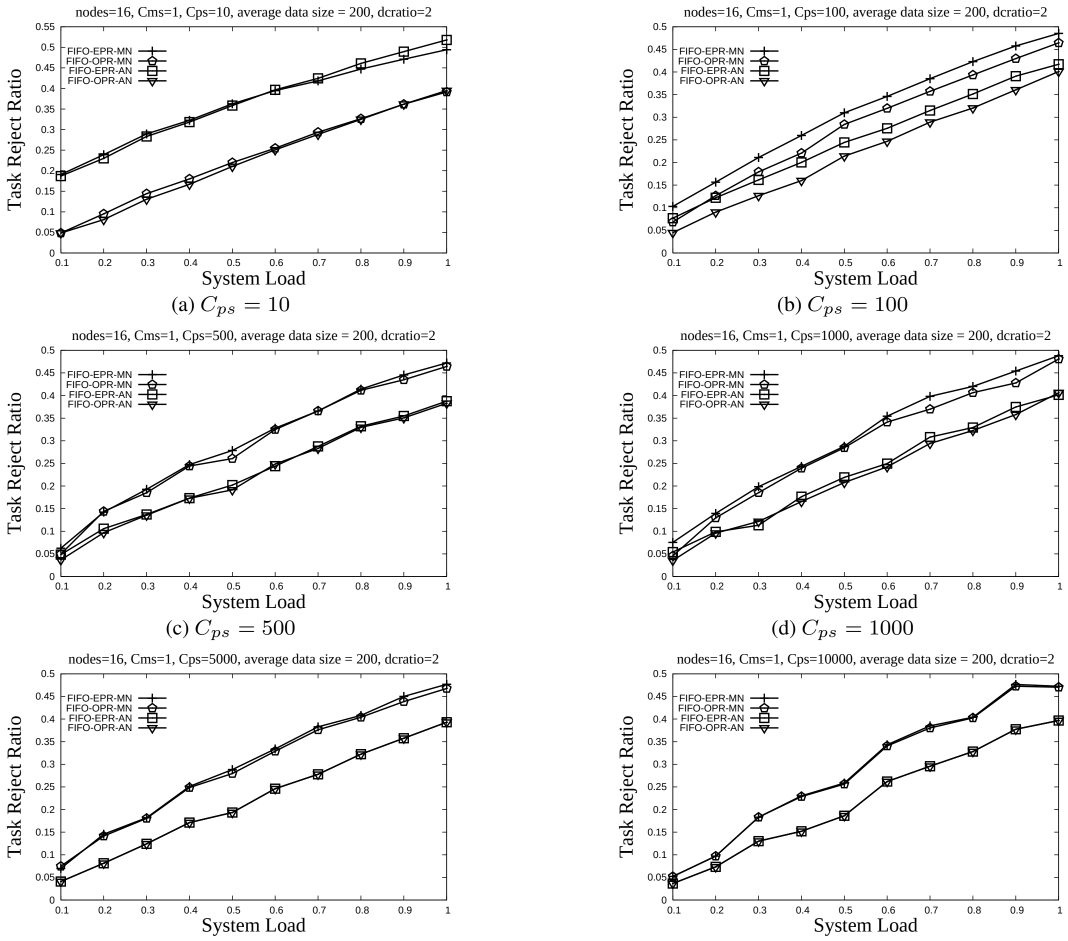
(e) $DCRatio = 3, C_{ms} = 1$

(f) $DCRatio = 10, C_{ms} = 1$

(g) $DCRatio = 20, C_{ms} = 1$

(h) $DCRatio = 100, C_{ms} = 1$

Figure 13: FIFO-OPR v.s. FIFO-EPR,$DCRatio = 2, 3, 10, 20, 100$



(a) $C_{ps} = 10$

(b) $C_{ps} = 100$

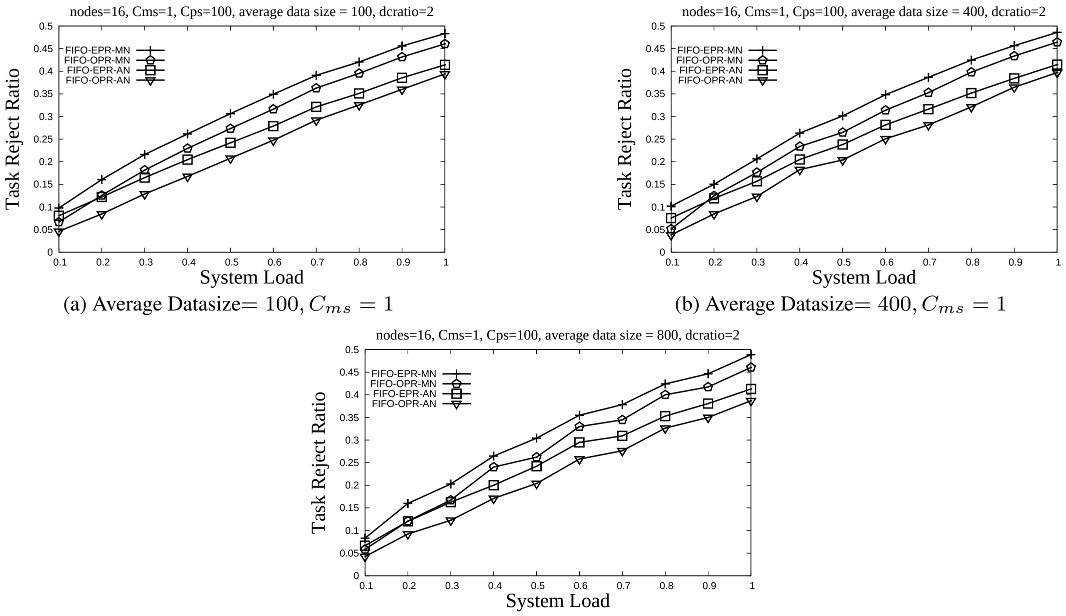
(c) $C_{ps} = 500$

(d) $C_{ps} = 1000$

(e) $C_{ps} = 5000$

(f) $C_{ps} = 10000$

Figure 14: FIFO-OPR v.s. FIFO-EPR,CPU

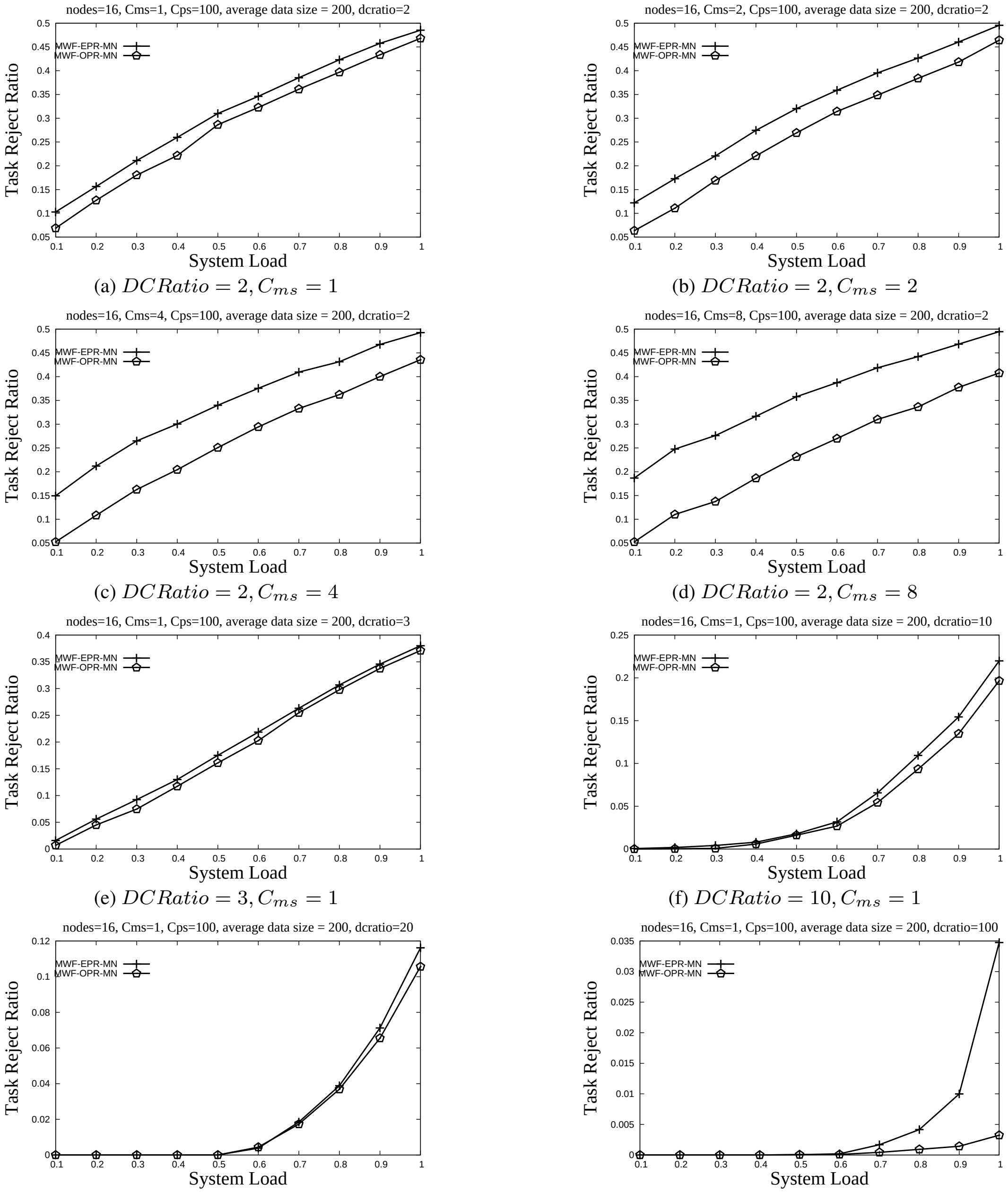


(a) Average Datasize$= 100, C_{ms} = 1$

(b) Average Datasize$= 400, C_{ms} = 1$

(c) Average Datasize$= 800, C_{ms} = 1$

Figure 15: FIFO-OPR v.s. FIFO-EPR, Average Datasize = 100, 400, 800



(a) $DCRatio = 2, C_{ms} = 1$

(b) $DCRatio = 2, C_{ms} = 2$

(c) $DCRatio = 2, C_{ms} = 4$

(d) $DCRatio = 2, C_{ms} = 8$

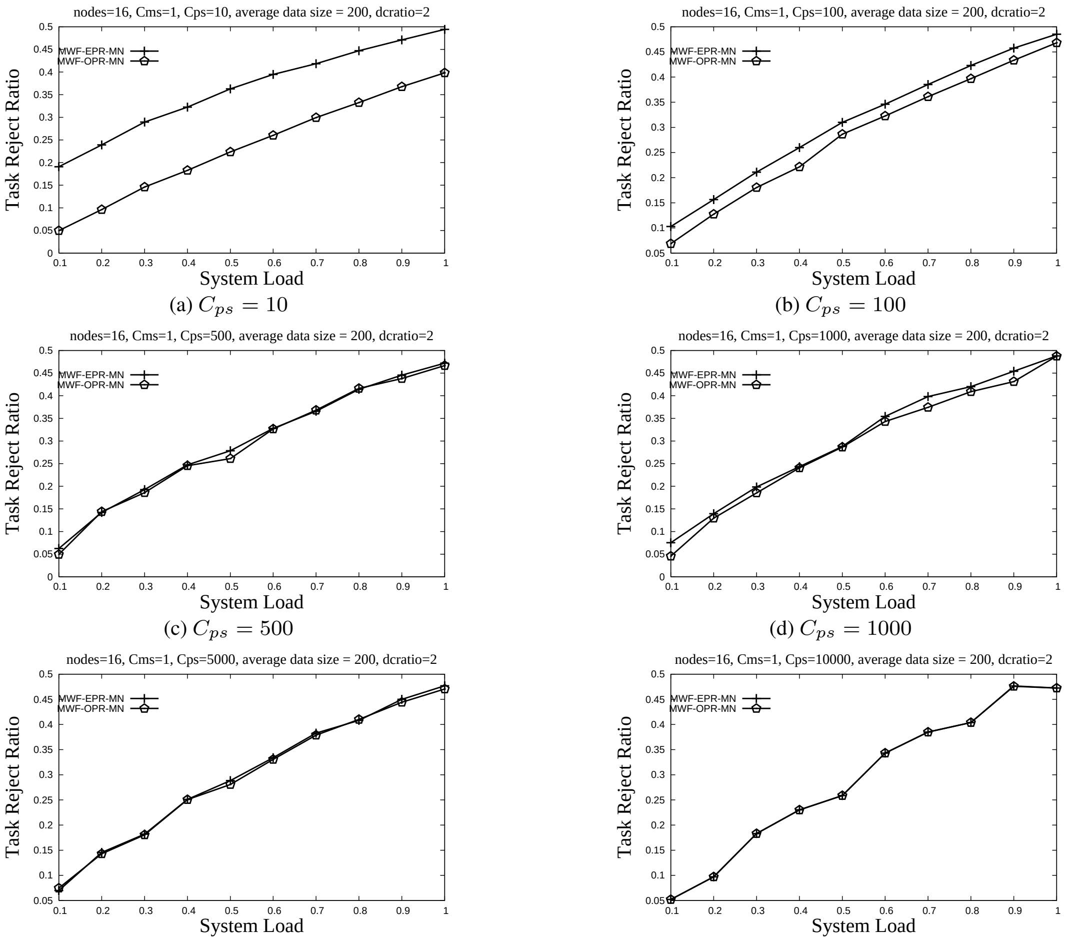
(e) $DCRatio = 3, C_{ms} = 1$

(f) $DCRatio = 10, C_{ms} = 1$

(g) $DCRatio = 20, C_{ms} = 1$

(h) $DCRatio = 100, C_{ms} = 1$

Figure 16: MWF-OPR v.s. MWF-EPR,$DCRatio = 2, 3, 10, 20, 100$



(a) $C_{ps} = 10$

(b) $C_{ps} = 100$

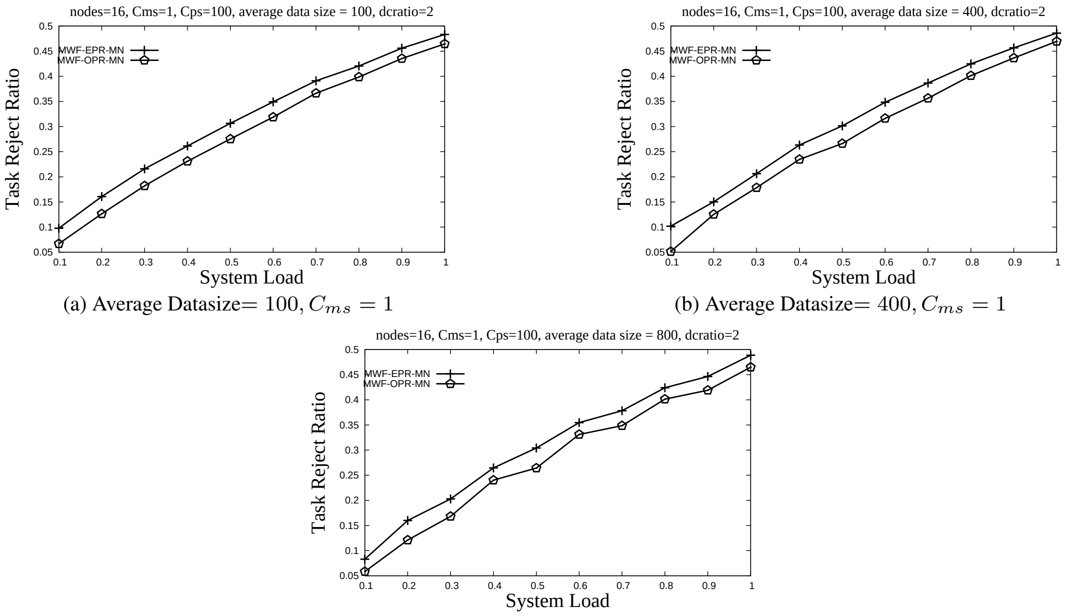
(c) $C_{ps} = 500$

(d) $C_{ps} = 1000$

(e) $C_{ps} = 5000$

(f) $C_{ps} = 10000$

Figure 17: MWF-OPR v.s. MWF-EPR,CPU

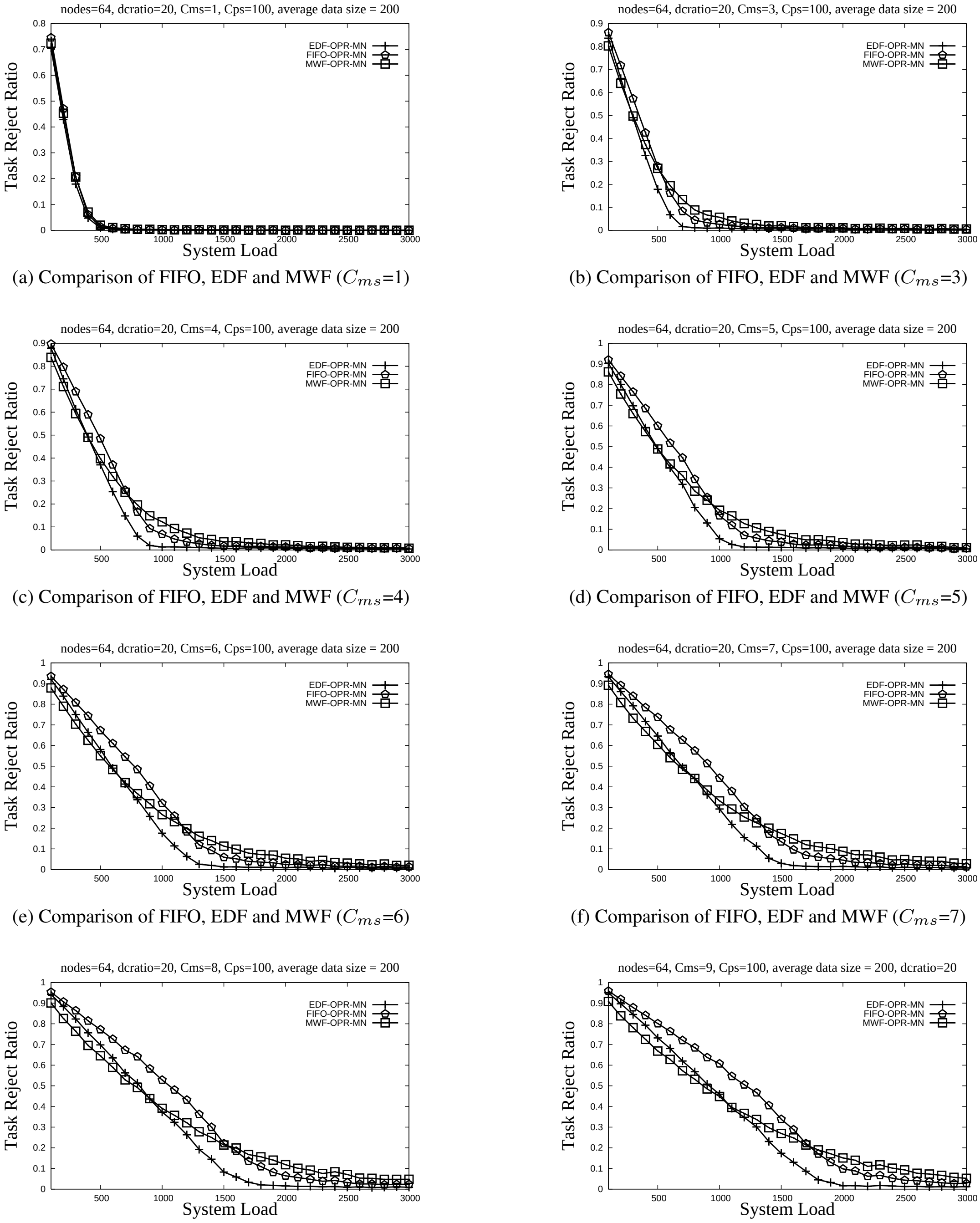


(a) Average Datasize$= 100, C_{ms} = 1$

(b) Average Datasize$= 400, C_{ms} = 1$

(c) Average Datasize$= 800, C_{ms} = 1$

Figure 18: MWF-OPR v.s. MWF-EPR, Average Datasize = 100, 400, 800



(a) Comparison of FIFO, EDF and MWF (C_{ms}=1)

(b) Comparison of FIFO, EDF and MWF (C_{ms}=3)

(c) Comparison of FIFO, EDF and MWF (C_{ms}=4)

(d) Comparison of FIFO, EDF and MWF (C_{ms}=5)

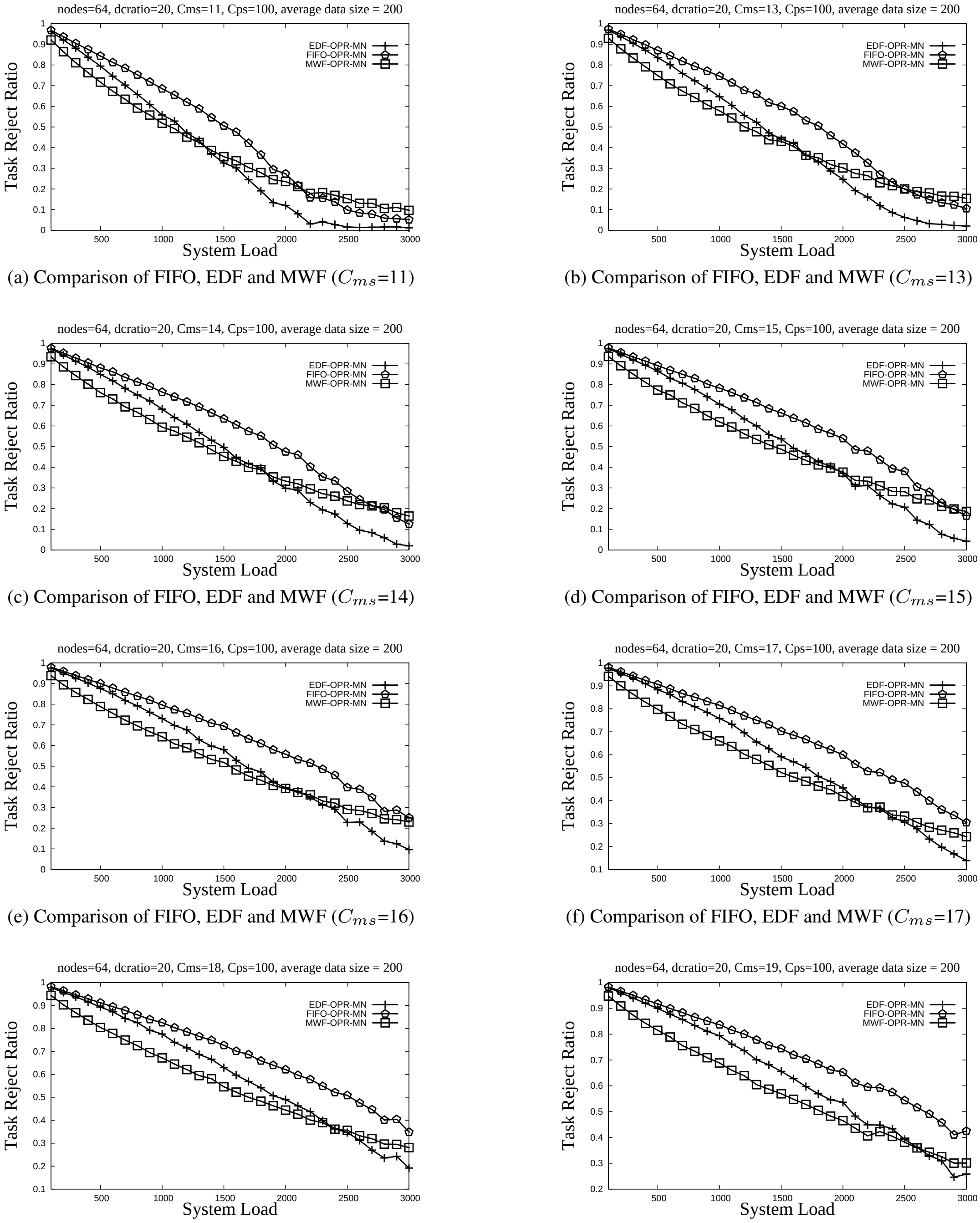
(e) Comparison of FIFO, EDF and MWF (C_{ms}=6)

(f) Comparison of FIFO, EDF and MWF (C_{ms}=7)

(g) Comparison of FIFO, EDF and MWF (C_{ms}=8)

(h) Comparison of FIFO, EDF and MWF (C_{ms}=9)

Figure 19: FIFO, EDF, MWF.



(a) Comparison of FIFO, EDF and MWF (C_{ms}=11)

(b) Comparison of FIFO, EDF and MWF (C_{ms}=13)

(c) Comparison of FIFO, EDF and MWF (C_{ms}=14)

(d) Comparison of FIFO, EDF and MWF (C_{ms}=15)

(e) Comparison of FIFO, EDF and MWF (C_{ms}=16)

(f) Comparison of FIFO, EDF and MWF (C_{ms}=17)

(g) Comparison of FIFO, EDF and MWF (C_{ms}=18)

(h) Comparison of FIFO, EDF and MWF (C_{ms}=19)

Figure 20: FIFO, EDF, MWF.