A Simulation Study of Data Partitioning Algorithms for Multiple Clusters

Chen Yu, Dan C. Marinescu*
School of Computer Science, University of Central Florida
Orlando, FL, 32816, USA. Email: (yuchen, dcm)@cs.ucf.edu
Howard Jay Siegel
Departments of Electrical & Computer Engineering and Computer Science
Colorado State University, Fort Collins, CO 80523-1373, USA. Email:HJ@colostate.edu
John P. Morrison
Computer Science Department, University College of Cork,
Cork, Ireland. Email: j.morrison@cs.ucc.ie

***Abstract*— Recently we proposed algorithms for concurrent execution on multiple clusters [9]. In this case, data partitioning is done at two levels; first, the data is distributed to a collection of heterogeneous parallel systems with different resources and startup time, then, on each system the data is evenly partitioned to the available nodes. In this paper, we report on a simulation study of the algorithms.**

I. Introduction

The Single Program Multiple Data (SPMD) paradigm [3] has been used for data-intensive applications for many years. The basic idea of this approach is to partition the data into several segments and then run the same program on several processors with a different segment of the data as input.

Often, the SPMD paradigm requires co-scheduling because processes communicate with each other; the members of a process group may need to carry out a barrier synchronization to make sure that all have completed a computation stage before proceeding to the next. Thus, they must be scheduled to run concurrently. Most co-scheduling algorithms are designed for homogeneous computing environments and are not suitable for large-scale distributed computing on a Grid. Several co-scheduling algorithms for a heterogeneous environment are based upon the assumption that the computation on each site can start at the same time [1]. This assumption is unrealistic for large-scale distributed computing when individual systems are located within different administrative domains and thus are autonomous.

There are "pleasantly" parallel algorithms where there is no communication among the members of the process group and we only need to merge the partial results once every member of the group finishes execution on its input data segment. Such "pleasantly" parallel problems are the focus of [9]. We assume that the programs running on heterogeneous parallel systems produce identical results, thus are logically equivalent to each other, though their implementations may differ, each may be optimized for the particular architecture of the target system. This extended computational model called GSPMD (Grid SPMD) requires more sophisticated algorithms for data partitioning among several clusters or massively parallel systems, and for scheduling on each system. Indeed, space sharing of a single cluster ensures that all nodes available for a given computation start processing at the same time, while this is no longer true for multiple clusters that may become available at different time.

The work we report in this paper is tied to our effort to build an intelligent environment for large-scale distributed computing applied to computational structural biology [2], [9]. The interest in the problem addressed in this paper is motivated by real-life applications. For example, we report that the computing time required to improve the resolution of a medium-sized virus such as Mammalian Reovirus (MRV), from about 7.6 Å to better than 7.0 Å on 42 processing nodes, is about 14 hours/iteration [5]. The refinement process in this case required about 100 iterations, thus the total time taken to improve the resolution from 7.6 Å to 7.0 Å was about $1,400$ hours, or nearly 60 days.

Sobering statistics such as these reflect that indeed we would greatly benefit if we could use concurrently multiple clusters. The cost of very large systems consisting of thousands of nodes is prohibitive for many research organizations, and the access to shared resources with very large clusters is limited. The structural biology problem mentioned earlier, is not unique, increasingly

* To whom correspondence should be addressed. Fax:(407)823-5419, Tel:(407)823-4860.

more applications tend to look at phenomena at molecular or atomic level and require a very large amount of computing cycles.

II. System Model and Basic Assumptions

A *process group* is a set of processes running concurrently on a set of nodes of a parallel system, be it a cluster, a PC farm, or a supercomputer; when the cardinality of a process group is 1, we have the traditional sequential execution on a single processor system. In this paper, we consider computations carried out by several process groups concurrently on multiple parallel systems. The fact that multiple process groups run concurrently does not mean that all must start at the same time.

We assume that an application $\mathcal{A}$ involves one computation $\mathcal{C}$ consisting of m process groups, all of which execute the logically equivalent programs, but each on a different data segment: $\mathcal{C} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots \mathcal{G}_m\}$. In turn, each process group $\mathcal{G}_i$ consists of n_i processes $\mathcal{G}_i = \{P_1^i, P_2^i, \ldots P_{n_i}^i\},\ 1 \leq i \leq m$. We assume that we have $n \geq m$ target systems that form the *target systems set* $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2 \ldots \mathcal{S}_n\}$ and wish to assign each process group to one target system for execution with the assumption that one target system will only accept one process group of current computation $\mathcal{C}$. We distinguish an application $\mathcal{A}$ from a computation $\mathcal{C}$ implementing the application on a particular system because the implementations may differ; the implementations produce identical results, thus are logically equivalent, but each may be optimized for the particular architecture of the target system.

The problem we wish to solve consists of several stages:

- Determine the value of parameters describing each system in the target systems set $\mathcal{S}$ for the computation $\mathcal{C}$ of application $\mathcal{A}$.
- Identify the *restricted target set*, $\mathcal{Q}^{(\pi)} \subseteq \mathcal{S}$ of size $|\mathcal{Q}^{(\pi)}| = m$ and determine the actual size of the data segment to be assigned to each restricted target system $\mathcal{S}_j \in \mathcal{Q}^{(\pi)}$. The restricted target systems in the restricted target set $\mathcal{Q}^{(\pi)}$ will be used to run the computation $\mathcal{C}$ and guarantee the shortest computation $\mathcal{C}$ completion time under the corresponding data partitions.
- Schedule the execution of computation $\mathcal{C}$ on restricted target systems with the assigned data segments.

An application is characterized by a *data unit (dtu)*, which reflects the logical organization of data for that application. For example, for the origin and orientation refinement [5], a dtu consists of a number of virus projections extracted from a micrograph. We define the *execution rate*, $\mu_j^{\mathcal{C}}$, of a computation $\mathcal{C}$ on a system $\mathcal{S}_j$, as the amount of data available locally, processed in one unit of time, using all of the resources on that system. The execution rate, measured in dtu per unit of time (seconds, minutes, hours), is a synergistic measure of performance that reflects a wide range of target system attributes, such as the system architecture, the CPU rate, the memory access time, the amount and speed of memory cache, the latency and the bandwidth of the interconnection network, and the I/O latency and bandwidth.

The execution rate also reflects the main attributes of the application, such as the size of the working data set, the type of parallelism (fine- versus course-grain), and whether the application is I/O-bound versus CPU-bound. For example, if we have a cluster with 128 nodes and the application is the parallel origin and orientation refinement, PO^2R, program [5], then we measure the time it takes to process 10^6 projections and if the time for one iteration is 5 hours, then we say the execution rate is 0.2×10^6 projections/hour. If we can use only half of the nodes, then the actual execution rate becomes 0.1×10^6 projections/hour.

The execution rate on one system determines the actual execution time for a particular application with a given input using all the resources on that system, thus, it reflects the user perception regarding the performance of a system for a particular computation. From this brief discussion, it should be clear that the execution rate can only be estimated experimentally and that it can be affected by contention for system resources (e.g., the bandwidth of the interconnection network, the I/O bandwidth) with other applications running concurrently, when a parallel system is space shared.

Consider two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ and two applications, $\mathcal{A}_1$ involving computation $\mathcal{C}_1$ and $\mathcal{A}_2$ involving computation $\mathcal{C}_2$. It is possible that $\mu_1^{\mathcal{C}_1} < \mu_2^{\mathcal{C}_1}$ but $\mu_1^{\mathcal{C}_2} > \mu_2^{\mathcal{C}_2}$. For example, consider the case when the application is the 3D-Discrete Fourier Transform (DFT) of a 3D-lattice of dimension l^3. The two parallel systems are: (a) distributed memory (*dm*) system, e.g., a cluster of PCs interconnected by a gigabit Ethernet, with a very large amount of memory per node and with a parallel file system, and (b) shared-memory (*sm*) system, e.g., a cluster of PCs with an omega interconnection network and a traditional file system. For the distributed memory system, we partition the data into slabs of width w_{dm}. The computation involves a 2D-DFT of all xy-planes of a slab along the z-axis, then a global exchange, and, finally, a 1D-DFT along the z-axis. For the shared-memory system, the data unit is a 3D-cube of dimension l_{sm}^3 such that $l_{sm}^3 = l^2 \times w_{dm}$ and there is no explicit communication among the processors as the data is in a shared memory and the kernel of the 3D-DFT carries out three 1D-DFTs. In this case, the execution rate of the

distributed memory system is most likely lower than that of the shared-memory system: $\mu_{dm}^{DFT} < \mu_{sm}^{DFT}$ because the algorithm is communication intensive (it requires all-to-all communication) and the latency and bandwidth of the gigabit Ethernet are no match for the interconnection network of the more expensive shared-memory system. On the other hand, if the application is a transaction processing (*tp*) system, a data unit consists of a number of transactions. It is likely that the parallel file system will favor the distributed memory system due to the faster parallel I/O system, thus, $\mu_{dm}^{tp} > \mu_{sm}^{tp}$. In this example, the codes for the DFT (and for the transaction processing system) are likely to be very different for the two systems, but for the same input, they should produce identical results.

The execution rate is static, it may change only over relatively long periods of time when the hardware and the software of a system is fixed. We assume that in addition to the static information provided by the execution rate, we have dynamic information regarding the current state of each target system. At time t, we expect to have information summarized by $\sigma^{\mathcal{C}}(t)$, the *startup vector*, indicating the expected time each target system will be available for computation $\mathcal{C}$, and by $\eta^{\mathcal{C}}(t)$, the *duty cycle vector*, indicating the expected fraction of resources available (i.e., based upon space-sharing of a multiprocessor system) for computation $\mathcal{C}$ on each target system: $\sigma^{\mathcal{C}}(t) = (\sigma_1^{\mathcal{C}}(t)\ \ \sigma_2^{\mathcal{C}}(t)\ \ \ldots\ \ \sigma_n^{\mathcal{C}}(t))$ and $\eta^{\mathcal{C}}(t) = (\eta_1^{\mathcal{C}}(t)\ \ \eta_2^{\mathcal{C}}(t)\ \ \ldots\ \ \eta_n^{\mathcal{C}}(t))$. For simplicity we shall drop the dependence of time and write $\sigma^{\mathcal{C}}$ and $\eta^{\mathcal{C}}$ instead of $\sigma^{\mathcal{C}}(t)$ and $\eta^{\mathcal{C}}(t)$, respectively.

Let ω be the input data size. An *allocation* of process groups $\mathcal{G}_i \in \mathcal{C}$ to systems $\mathcal{S}_j \in \mathcal{Q}^{(\pi)}$ is an one-to-one mapping $\nu : \mathcal{G}_i \mapsto \mathcal{S}_j$. Given an allocation mapping ν, the *data partitioning problem* for ν is to compute a decomposition δ of the data set into segments of size $\omega_1^{\nu}, \omega_2^{\nu}, \ldots \omega_m^{\nu}$ such that $\omega = \sum_{i=1}^{m} \omega_i^{\nu}$.

The pair $\pi = (\nu, \delta)$ describing both the allocation of process groups to target systems and the data partitioning is called a *mapping* of $\mathcal{C}$ to $\mathcal{S}$. The *completion time vector* for $\mathcal{C}$ associated with a particular mapping π is $T^{(\mathcal{C},\pi)} = (T_1^{(\mathcal{C},\pi)}\ \ T_2^{(\mathcal{C},\pi)}\ \ \ldots\ \ T_m^{(\mathcal{C},\pi)})$, where $T_j^{(\mathcal{C},\pi)}$ denotes the process group computation completion time on the target system $\mathcal{S}_j$ under mapping π.

$\omega_j^{(\pi)}$ represents the size of a data segment assigned to the system $\mathcal{S}_j$ under mapping π. For the sake of simplicity, we shall drop the superscript $\mathcal{C}$ and write $T^{(\pi)}$ instead of $T^{(\mathcal{C},\pi)}$, or $\sigma(t)$ instead of $\sigma^{\mathcal{C}}(t)$ whenever the context reveals that we are considering computation $\mathcal{C}$. The actual completion time of $\mathcal{C}$ under mapping π is $\tau^{(\pi)} = \max(T_1^{(\pi)}, T_2^{(\pi)}, \ldots T_m^{(\pi)})$. At time t, our goal is to determine the *optimal mapping*, π, which ensures the earliest completion time: $\min_{\pi}[\tau^{(\pi)} - t]$.

We use R_j, the *data transfer rate*, to measure the network transfer rate between the restricted target system $\mathcal{S}_j$ and the system where the input data set is located. Thus the data segment size that $\mathcal{S}_j$ can get in the time interval [t, $\sigma_j(t)$] is $(\sigma_j(t) - t)R_j$.

The basic assumptions of our model are:

1) We assume that during execution there is no communication among process groups. In other words, the first-level data partitioning needs not to consider the synchronization and communication between process groups. However, there may be communication among processes in one process group.
2) The heterogeneity does not affect the quality of the results. In other words, the partial results do not need any additional processing. The computation has a post-processing phase when partial results are merged together after all process groups have finished their execution. The time spent on post-processing phase does not count into computation time.
3) The information μ_j, σ_j, η_j and R_j regarding the state of system $\mathcal{S}_j$ is relatively stable; it does not change from the time when the data partitioning and the mapping are computed until the computation finishes its execution.

As usual, we use the speedup to measure the effectiveness of parallel execution. The relative improvement of mapping π_2 with completion time $\tau^{(\pi_2)}$ over mapping π_1 with completion time $\tau^{(\pi_1)} \geq \tau^{(\pi_2)}$ is measured by the relative speedup: $Speedup_{\pi_2/\pi_1} = \tau^{(\pi_1)}/\tau^{(\pi_2)}$.

III. Optimal Mapping and Data Partitioning Algorithms

An optimal mapping and data partitioning algorithm for a GSPMD problem ensures the earliest possible completion time given the time when each system becomes available, as well as the execution time on each system. Unfortunately, we cannot estimate very accuracy either the execution time, or the time when resources become available, thus in practice the mapping and data partitioning are likely to be near-optimal [9].

We call the algorithms for finding an optimal mapping without data staging:

- *Flexible Mapping (FlexMap)* - when we wish to use as many systems as possible to ensure the earliest possible completion time, and
- *Fixed Mapping (FixMap)* - when the number of systems is restricted to a specific value.

FlexMap algorithms are well suited when there are no restrictions regarding the maximum number of process groups, or the number of target systems we could use. A greedy algorithm that uses as many target systems as feasible has the shortest possible completion time. The basic idea of the *FlexMap* algorithm [9] is to order the

systems in target set $\mathcal{S}$ based on their startup times with the earliest first, then start with an empty restricted target set, iteratively add a new target system with the earliest startup time to it and then compute the new completion time τ until no improvement can be achieved.

In practice, we expect to be constrained either by the the internal logic of application which limits the number of process groups, or by considerations such as the cost to access a system, the complexity of coordination, or the failure rates. To address these problems, we developed *FixMap* algorithm [9].

For many applications the data staging time could be significant. While the nodes within one parallel system are often inter-connected by Gbps networks, multiple parallel systems are interconnected via the Internet and/or Local Area Networks where congestion control and contention limit the actual data transmission rates and lead to a significant data staging time.

We also consider a data partitioning scheme that ensures that data staging for each target system $\mathcal{S}_i$ in restricted target set $\mathcal{S}^{(\pi)}$ finishes before σ_i, the time when system $\mathcal{S}_i$ becomes available. We call the optimal mapping algorithms with data staging *FlexMapDS* and *FlexMapDS*, respectively.

IV. A Simulation Study

We simulate an ensemble of 200 target systems and investigate the completion time, τ, function of the input data set size, ω for *FlexMap* and *FlexMapDS* algorithms. We also study the completion time function of the number of process groups, m, for *FixMap* and *FixMapDS* algorithms.

Each target system is characterized by a vector consisting of: the startup time σ, the execution rate μ, the duty cycle η, and the data transfer rate R. The four random variables are normally distributed. The mean and standard deviations of the four random variables are respectively: σ (the start-up time) 25 and 5 hours; μ (the execution rate) 750 and 80 Mdtu/hour; η (the duty cycle) 0.75% and 0.05%; and finally R (the data transfer rate) 450 and 30 Mdtu/hour.

In our simulation we first construct n random vectors $(\mu_i, \sigma_i, \eta_i, R_i), \quad \forall \mathcal{S}_i \in \mathcal{S}$ which characterize the target set $\mathcal{S}$. Then the algorithms select the restricted target set, $\mathcal{Q}^{(\pi)}$, and the amount of data allocated to each system. The algorithms are deterministic, thus for a given target set configuration, $\mathcal{S}$, and input data set size, ω, the completion time and the number of systems in the restricted target set, $\mid \mathcal{Q}^{(\pi)} \mid$, are the same under mapping π during each run.

For some of the experiments, we calculate the confidence intervals for different target set configurations. Multiple target set configurations are derived from a basic configuration by shuffling the process rates on target systems, while keeping the other parameters, the startup time, the duty cycle, and the data transfer rate unchanged. We generate 100 different configurations, run the algorithms on each configuration, and record the 95% confidential intervals.

A. FlexMap and FlexMapDS

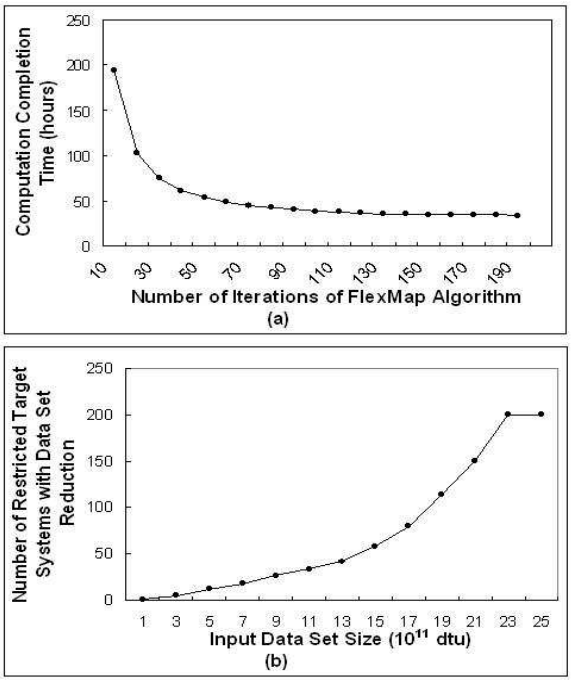

Fig. 1. (a) *FlexMap* algorithm. The completion time (in hours) function of the number of iterations for a fixed input data set size. At each iteration we add a new system to the restricted target set if the startup time of the system is earlier that the current completion time; thus, the completion time monotonously decreases as we iterate. The input data size $\omega = 10^{12}$ dtu. (b) *FlexMapDS* algorithm. The number of target systems which experience data set reduction to guarantee the termination of data staging, function of the input data set size (in units of 2×10^{11} dtu).

We study the effect of the input data set size (measured in dtu, data units) upon the completion time and upon the optimal number of systems used. For the first type of studies of the *FlexMap* algorithm we consider a fixed input data set size, $\omega = 10^{12}$ *dtu*; the algorithm terminates after 191 iterations, thus we could use for data partitioning 191 systems out of the 200 systems available. Figure 1(a) shows the completion time function of the number of iterations and Figures 2(a) and 2(b) detail the first 15 and the last 15 iterations, respectively. If we define the speedup as the ratio of the completion time for the first iteration of a range of consecutive iterations to the completion time for the last iteration of the range, we notice that:

$$Speedup_{1-15} = \frac{2050}{100} = 20.50 \ , \qquad (1)$$

$$Speedup_{177-191} = \frac{34}{33.9} = 1.00294. \qquad (2)$$

This indicates to us that the benefits of using an increasingly larger number of systems have to be balanced against the additional cost and overhead to coordinate multiple sites.

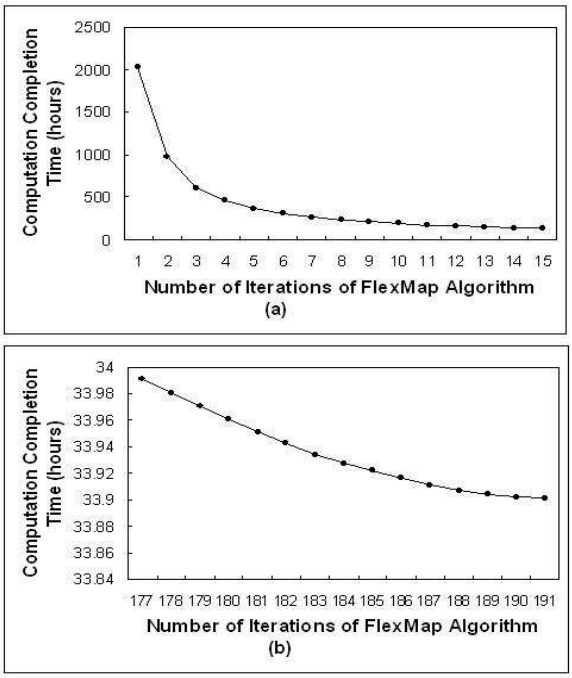

Fig. 2. *FlexMap* algorithm. (a) The completion time (in hours) during the first 15 iterations. (b) The completion time (in hours) during the last 15 iterations of the execution simulated in Figure 1(a).

In fact, the *FlexMap* algorithm should have an additional termination condition:

FlexMap Algorithm With Controlled Speedup

.....
if $(\tau^{(j)}/\tau^{(j-1)} < MinSpeedupPerIteration)$
exit;
.....
End

To study the effect of the data set size, we increase ω from 10^{11} dtu to 25×10^{11} dtu in increments of 2×10^{11} dtu and observe the evolution of the completion time, Figure 3(a), and the size of the optimal target systems set, Figure 3(b).

Figure 3(b) reveals that, as expected, the number of systems used for a given application increases as the input data set size increases for the *FlexMap* and *FlexMapDS* algorithms, given an input target system set $\mathcal{S}$. In this experiment, the resource vectors $(\mu_i, \sigma_i, \eta_i, R_i),\ \forall \mathcal{S}_i \in \mathcal{S}$ are the same, only the data set size increases. For a relatively small input data

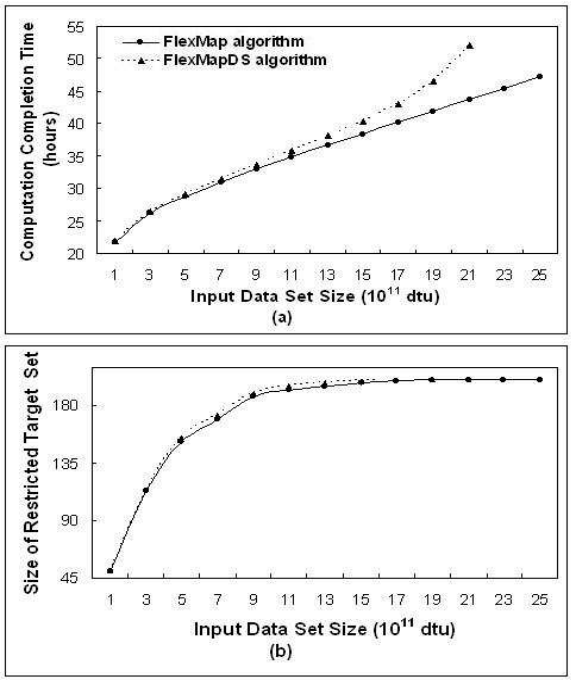

Fig. 3. *FlexMap* and *FlexMapDS* algorithms. (a) The completion time (in hours) function of the input data set size; the input data set size increases in units of 2×10^{11} dtu. (b) The size of restricted target set (the number of systems used) when the input data set size increases in units of 2×10^{11} dtu.

set size, after a few iterations, the completion time becomes shorter than the startup time of the systems outside of the restricted target set (the systems we have already selected). As the input data set size increases, the completion time after the same number of iterations increases and allows us to include more systems in the restricted target set. We conclude that:

$$\omega_a > \omega_b \implies |\ \mathcal{Q}^{(\pi)}(\omega_a)\ | \ \geq \ |\ \mathcal{Q}^{(\pi)}(\omega_b)\ |, \quad (3)$$

and a similar relation holds for the completion time after iteration j:

$$\omega_a > \omega_b \implies \tau^{(j)}_{\omega_a} \geq \tau^{(j)}_{\omega_b}. \qquad (4)$$

For example, consider the *FlexMap* algorithm; when $\omega = 10^{11}$ dtu, only 50 systems could be included in the restricted target set, the algorithm stops after 50 iterations; the completion time is 21.7 hours. As the amount of input data, ω, increases to 3×10^{11} dtu, an additional 63 target systems are included in the restricted target set and the algorithm stops after 113 iterations; the completion time becomes 26.1 hours. Thus, even though the data set size doubles, the completion time increase only by 25%. It is also interesting to note that the number of systems targeted by the algorithm more than doubles, thus the amount of data allocated to the first 50 systems increases only by about 20%. We expect

that:

$$\frac{n}{m} >> 1 \implies \frac{\omega_a}{\omega_b} >> \frac{\tau^{(\pi)}_{\omega_a}}{\tau^{(\pi)}_{\omega_b}}. \quad (5)$$

In this experiment, when the input data set increases 19 fold, $\omega = 19 \times 10^{11}$ dtu, the *FlexMap* algorithm is able to use all $n = 200$ systems in the target set. As the number of systems in the restricted target set approaches n, it becomes increasingly more difficult to find systems that can be included in the restricted target set, Figure 3(b).

The completion time computed with the *FlexMapDS* algorithm is slightly longer than the one produced by the same algorithm without data staging, Figure 3(a); the difference increases as the input data size increases. This behavior is expected; to ensure that data staging is completed before the actual startup time of a system, the algorithm may reduce the amount of input data assigned to some systems in the restricted target set, and thus delay the completion time.

Figure 3(b) shows that the size of the restricted target set becomes slightly larger when input data set size increases and data staging is involved. This is due to the fact that more systems in the restricted target cannot finish their data staging in time based on its original data partition produced by the *FlexMap* algorithm; the algorithm has to reduce the amount of data allocated to these systems, Figure 1(b). When the input data set size reaches 23×10^{11} dtu, the *FlexMapDS* algorithm does not produce a solution; the amount of data allocated to each one of the 200 target systems is reduced and the total amount of data that can be processed is smaller than the input data set size ω.

B. FixMap and FixMapDS

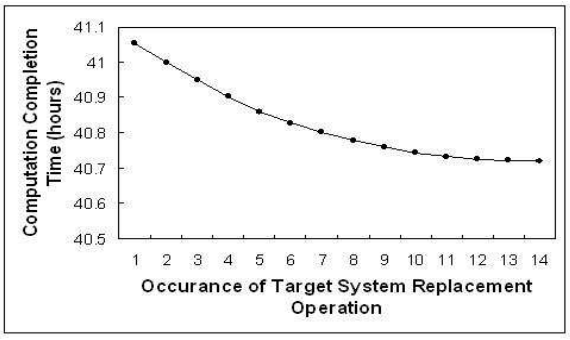

Fig. 4. *FixMap* algorithm. The completion time (in hours) after each target system replacement operation; $\omega = 10^{12}$ dtu, $m = 85$. 95% confidence intervals are shown.

The *FixMap* algorithm is more complex, it requires occasional reorganization of the restricted target set and occasional removal from this set of less performant systems. For our first experiment, the data set size is $\omega = 10^{12}$ dtu and the size of the restricted target set is limited to $m = 85$. We record the completion time under the *FixMap* algorithm when a system from the restricted target set is replaced by a more performant one. Figure 4 shows that 14 such replacement operations lead to a speedup of:

$$Speedup_{14\ replc} = \frac{41.05}{40.71} = 1.008. \quad (6)$$

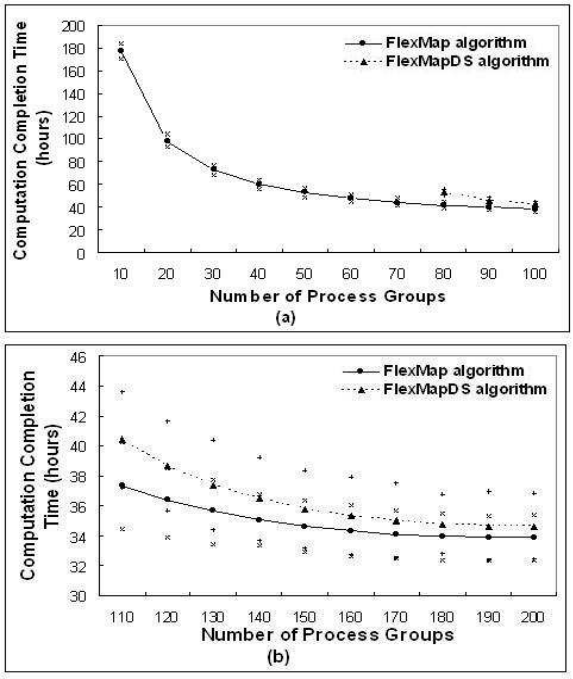

Fig. 5. *FixMap* algorithm. The completion time (in hours) when (a) $10 \leq m \leq 100$; $\omega = 10^{12}$ dtu; (b) $110 \leq m \leq 200$, $\omega = 10^{12}$ dtu. 95% confidence intervals are shown.

Next, we study the speedup of the *FixMap* and *FixMapDS* algorithms when the number of process groups increases, but the input data set size is fixed, $\omega = 10^{12}$ dtu. Figures 5 (a) and (b) present the case when $10 \leq m \leq 100$ and $110 \leq m \leq 200$, respectively. The speedup due to the *FixMap* algorithm is:

$$Speedup_{10-100} = \frac{180}{40} = 4.5\ , \quad (7)$$

$$Speedup_{110-200} = \frac{37.4}{33.9} = 1.10324. \quad (8)$$

The results are similar to the ones obtained for the same input data set size, $\omega = 10^{12}$ dtu) with ther *FlexMap* algorithm, Figure 2. The *FixMapDS* algorithm produces no solution when $m < 80$; the total amount of data processed by a subset consisting of 80 or fewer systems is smaller than the input data set size 10^{12} dtu. When $m \geq 80$, the computation completion time generated by *FixMapDS* algorithm is longer than the

counterpart generated by *FixMap* algorithm. This result is expected because some restricted target systems may experience data set reduction, when data staging is considered, in order to finish their data staging on time, and thus lead to the redistribution of workload and a longer computation completion time. Note that the gap between completion times generated by the *FixMapDS* and *FixMap* algorithms keeps decreasing as m increases. This tendency implies that the discrepancies between the restricted target sets generated by the *FixMapDS* and *FixMap* algorithms become less significant when m increases. Indeed, as the amount of data assigned to each target system is smaller when m increases, the chance that the data staging does not finish on time decreases. The *FixMapDS* and *FixMap* algorithms produce identical results when all the systems in the restricted target set generated by the *FixMapDS* algorithm finish their data staging before their startup time.

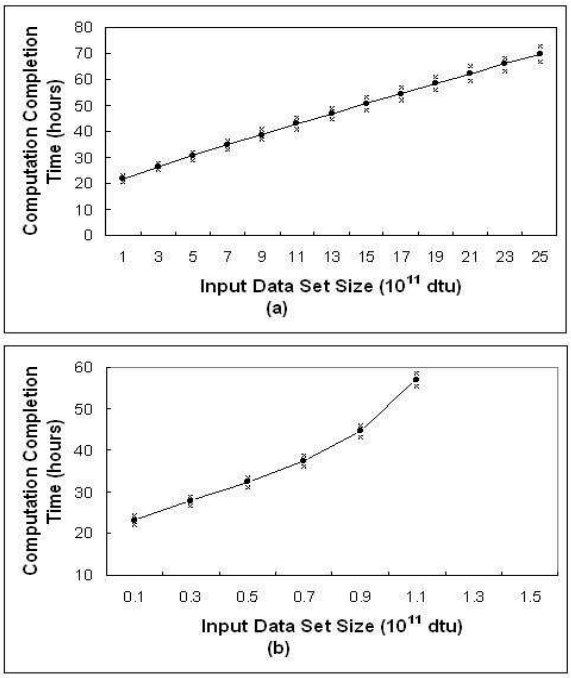

Fig. 6. (a) *FixMap* algorithm. The completion time (in hours) function of the input data set size; the input data set size increases in units of 2×10^{11} dtu and $m = 85$. (b) *FixMapDS* algorithm. The input data set size increases in units of 0.2×10^{11} dtu and $m = 85$. 95% confidence intervals are shown.

Next, we investigate the effect of the input data set size on the results produced by the and *FixMapDS* algorithms. For *FixMap* we set $m = 85$ and increase the data set size in the range $10^{11} \leq \omega \leq 25 \times 10^{11}$ dtu in steps of 2×10^{11} dtu; for *FixMapDS* we increase the data set size in steps of 0.2×10^{11} dtu, in the range $0.1 \times 10^{11} \leq \omega \leq 1.5 \times 10^{11}$ dtu, Figure 6(a). The completion time increases linearly with the size of the input data set for the *FixMap* algorithm. Since m is fixed, the linear increase shows that the data partitioning algorithm works well; the additional amount of data allocated to each system is proportional to the system resources.

Figure 6(b) shows that the *FixMapDS* algorithm does not produce a solution when $\omega > 1.1 \times 10^{11}$ dtu. The computation completion time increases linearly at first, and, as the input data size keeps growing, it increases much faster. When the amount of input data is relatively small, few systems cannot finish data staging on time and experience a data set reduction; also, the pool of systems outside the restricted target set is large and these systems can be used to reduce the completion time and overcome the effect of larger input data size. When the input data size is large and the number of process groups is fixed, a larger percentage of systems fail to finish data staging in time; then the algorithm has to reduce the data segment size assigned to them but fewer systems are in the pool of potential replacements.

Figures 6(a) and 3(a) show that when input data set size is relatively small the *FixMap* and FlexMap algorithms exhibit similar behavior. When $\omega = 10^{11}$ dtu, although $m = 85$, only 50 target systems could be used; the explanation for this behavior was given in Section IV-A - Figure 3 (b) - and completion time was the same as the one using the *FlexMap* algorithm. When ω increases to 3×10^{11} dtu, the *FlexMap* algorithm uses 113 systems while the *FixMap* algorithm could only use 85 systems. The *FlexMax* algorithm always ensures the shortest possible completion time for a given data set size and for a given configuration of available systems; this is confirmed by the simulation results when $\omega > 3 \times 10^{11}$ dtu.

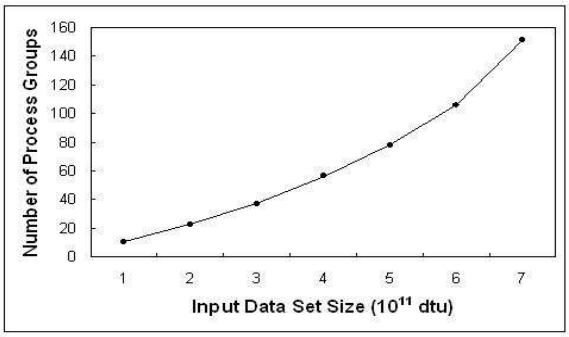

Fig. 7. *FixMap* algorithm. The number of process groups function of the input data set size to achieve the same completion time; initially $\omega = 10^{11}$ dtu and $m = 10$.

Lastly, we investigate if the *FixMap* algorithm allows us to maintain the same completion time when the input data set size increases. Figure 7 illustrates the evolution of the number of process groups (equal to the number of systems in the restricted target set) function of the

input data set size, when we force the completion time to be constant. Initially $\omega = 10^{11}$ dtu and $m = 10$; for $\omega = 5 \times 10^{11}$ dtu, we have $m = 80$. If $\omega \geq 8 \times 10^{11}$ dtu, we can no longer ensure the same completion time with 200 target systems.

V. Summary and Future Work

The time complexity of the algorithms presented in this paper are: $\mathcal{O}(n \log(n))$ for *FlexMap*, $\mathcal{O}(n^2)$ for *FixMap*, $\mathcal{O}(n^2 \log(n))$ for *FlexMapDS*, and $\mathcal{O}(n^3 \log(n))$ for *FixMapDS* [9]. While the efficiency of the algorithms is of concern, we note that in practice the number of systems in the target set, n, is relatively small, unlikely to exceed 10^3, thus the running time of the mapping and data partitioning algorithms are likely to be of the order of seconds. Yet algorithm efficiency is critical for error recovery.

Our main concern is the speedup of the data-intensive computation, and this depends upon the total amount of computing cycles available. An optimal mapping and data partitioning algorithm ensures the earliest possible completion time given the time when each system becomes available, as well as the execution time on each system. Unfortunately, we cannot estimate with utmost accuracy either the execution time, or the time when resources become available, thus in practice the mapping and data partitioning are likely to be near-optimal.

FlexMap algorithms always ensure the earliest possible completion time because, in principle, they can take advantage of all the systems available, while *FixMap* algorithms are required to use at most a subset of size m of all the n systems available. Once we fix n, the completion time initially decays exponentially as the number of systems used increases for both *FlexMap* and *FixMap* algorithms. As we approach saturation, when $m \to n$, the speedup obtained by increasing the number of systems used, becomes closer to 1. As we note in Section IV-A, the benefits of using an increasingly larger number of systems have to be balanced against the additional cost and overhead to coordinate multiple sites and the *FlexMap* algorithm should have an additional termination condition reflecting a cost-benefit analysis.

When the data set size increases the algorithms expand the set of systems to which process groups are mapped, initially faster, and, as $m \to n$, this expansion proceeds at a much slower rate because fewer systems are available. The completion time increases linearly for a *FixMap* algorithm and quasi-linearly for a *FlexMap* one. If we wish to keep the same completion time when the input data set size increases we need to use an increasingly larger number of systems.

The algorithms are extended to cover the case when the data staging cannot be completed before each system becomes available. In this case, we determine an approximate data staging time for each site. Then we adjust the amount of data allocated to each system to guarantee that the data staging terminates before the system becomes available. Our simulation shows that data staging has little effect, it increases slightly the completion time and the size of the restricted target set.

VI. Acknowledgements

This research was supported in part by National Science Foundation grants ACI0296035, EIA0296179, CNS0615170, the Colorado State University George T. Abell Endowment, and by I2Lab Graduate Fellowship at the College of Engineering & Computer Science at University of Central Florida.

References

[1] M.J. Atallah, C.L. Black, D. C. Marinescu, H.J. Siegel, and T.L. Casavant. *Models and Algorithms for Co-Scheduling Compute-Intensive Tasks on a Network of Workstations*. J. Parallel and Distrib. Comp., **16**(4):319–327, 1992.

[2] X. Bai, H. Yu, G. Wang, Y. Ji, D. C. Marinescu, G.M. Marinescu, and L. L. Bölöni. *Coordination in Intelligent Grid Environments*. Proc. IEEE, **93**(3):613–630, 2005.

[3] F. Darema-Rodgers, V.A. Norton, and G.F. Pfister. *Using A Single-Program-Multiple-Data Computational Model for Parallel Execution of Scientific Applications*. Technical Report RC11552, IBM T.J Watson Research Center, November 1985.

[4] N. Fujimoto and K. Hagihara. *Near-optimal Dynamic Task Scheduling of Independent Coarse-grained Tasks onto a Computational Grid.* 32th Int. Conf. on Parallel Processing (ICPP-03), pp. 391–398, 2003.

[5] Y. Ji, D.C. Marinescu, W. Zhang, X. Zhang, X. Yan, and T.S.Baker. A Model-based Parallel Origin and Orientation Refinement Algorithm for CryoTEM and its Application to the Study of Virus Structures. *J. Struct Biology*, **154** (1):1–19, 2006.

[6] H.D. Karatza. *A Simulation - Based Performance Analysis of Gang Scheduling in a Distributed System.* Proc. 32nd Simulation Symp., pp. 11–15, 1999.

[7] K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak and J. Pukacki. *Improving Grid Level Throughput Using Job Migration And Rescheduling*. Scientific Programming **12**(4):263–273, 2004.

[8] S.Y. You, H.Y. Kim, D. H. Hwang, and S.C. Kim. *Task Scheduling Algorithm in GRID Considering Heterogeneous Environment*. Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications, (PDPTA '04), pp. 240–245, 2004.

[9] C. Yu, G., D. C. Marinescu, H.J. Siegel, and J. P. Morrison. *Data Partitioning for Large-Scale Distributed Systems* (submitted).