The Impact of I/O on Program Behavior and Parallel Scheduling

Emilia Rosti[*], Giuseppe Serazzi[†], Evgenia Smirni[‡], Mark S. Squillante[§]

[*] Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Milano, Italy. rose@dsi.unimi.it

[†] Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy. serazzi@elet.polimi.it

[‡] Department of Computer Science, College of William and Mary, Williamsburg, VA, USA. esmirni@cs.wm.edu

[§] IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, USA. mss@watson.ibm.com

Abstract

In this paper we systematically examine various performance issues involved in the coordinated allocation of processor and disk resources in large-scale parallel computer systems. Models are formulated to investigate the I/O and computation behavior of parallel programs and workloads, and to analyze parallel scheduling policies under such workloads. These models are parameterized by measurements of parallel programs, and they are solved via analytic methods and simulation. Our results provide important insights into the performance of parallel applications and resource management strategies when I/O demands are not negligible.

1 Introduction

Parallel computing has emerged as an indispensable tool for solving problems in various scientific/engineering domains, as many classical disciplines are increasingly dependent upon powerful parallel systems for the execution of both computation and data intensive applications. The allocation and management of resources in these system environments is a particularly complex problem that is fundamental to sustaining and improving the benefits of parallel computing. The large number of users attempting to simultaneously execute work on the system, the parallelism of the applications and their respective computational and secondary storage needs, the satisfying of execution deadlines for certain applications, are all issues that exacerbate the resource scheduling problem.

Most of the previous research in this area has focused on the scheduling of processors in parallel systems. This includes various forms of space-sharing where the processors are partitioned among different parallel jobs (e.g., [5, 2, 12, 16]), and to a lesser extent combinations of space-sharing and time-sharing (e.g., [7, 18, 3]). Some studies have also considered the coordinated allocation of processors and memory [9, 13, 8]. However, an important dimension that has been essentially ignored in previous parallel scheduling research concerns the impact of the I/O demands of the parallel applications.

For many applications, and specifically those in the computational science and engineering fields, manipulating large volumes of data is a necessity and high-performance I/O becomes critical for performance. Some early characterizations of I/O-intensive parallel applications suggest that the I/O requirements of a parallel job can significantly affect the job efficiency and scalability [15]. The application scalability is in fact a function of both the number of assigned processors and the configuration of the I/O system (i.e., the number of available disks and the distribution of the application data on the disks). Furthermore, if the application has to compete with other jobs for secondary storage resources, its efficiency may decrease due to the increase in time spent by the application waiting for its I/O requests to complete. As a result, there may be significant perturbations in the expected I/O service times and consequently in the job execution times.

The problem of coordinated allocation of processor and disk resources has two fundamental aspects that are strongly interconnected. One concerns the behavior of parallel programs and workloads with respect to computation and I/O as a function of the number of processors and disks on which they are executed. The other concerns the allocation and management of resources among the parallel jobs comprising the system workload to satisfy various objective functions, such as minimizing response time and maximizing throughput. In this paper we develop a systematic approach to investigate these key issues of the general resource scheduling problem in parallel systems, and we exploit our formal framework to gain a better understanding of this complex problem.

First, we formulate a detailed yet compact model of parallel program behavior that effectively captures the I/O and computation characteristics of parallel applications which we observe from measurement data obtained as part of this study. A functional form for application speedup with respect to both processor and disk resources is derived, leading to a generalization of Amdahl's law [1] that extends the function to three dimensions to also include the disks. This analysis is then extended to address the computation and I/O behavior of parallel workloads. The models and analysis are validated against measurements of parallel applications and shown to be in excellent agreement. Our analysis is used to examine various performance measures of parallel applications as a function of the number of processors and disks assigned to them. In addition to this formal analysis, our methodology provides several practical benefits that include significantly more accurate predictions than from measurements alone and considerable simplifications in acquiring (as well as more accurate) measurements.

We then exploit our formal models and analysis of parallel program and workload behavior as the foundation for our analysis of coordinated processor and disk resource allocation strategies. A set of scheduling policies is evaluated, which spans the range from simple space-sharing schemes to combinations of space-sharing and time-sharing where the I/O requests of some jobs are overlapped with the computation

This work was partially supported by the PQE2000 project and the Italian CNR CESTIA center. Portions of this research were conducted during an academic visit by M. S. Squillante to the Dipartimento di Elettronica e Informazione, Politecnico di Milano.

of other jobs. Our formal analysis includes the derivation of a generalization of standard flow-equivalent service centers to capture the contention for disk resources in our analytic scheduling models. Our results demonstrate that when there is contention for the disk resources, the trends in system performance can be significantly different from those appearing in the research literature when I/O behavior is negligible or not explicitly considered. In particular, the trends for the best partition size under static space-sharing do not decrease with increasing system load in most cases, and many of the static policies outperform a favorable version of dynamic partitioning (i.e., no reconfiguration overhead). We show that the I/O behavior of parallel workloads can have a dominating effect on system performance under various resource allocation policies in a manner somewhat analogous to earlier results demonstrating that memory overhead can dominate the execution time of a parallel job when it is allocated too few processors [9, 13, 8]. Our results further show the potential for significant performance benefits by combining space-sharing with time-sharing to overlap the I/O of some jobs with the computation of other jobs.

§2 presents our models and analysis of parallel program and workload behavior. §3 presents our models and analysis of several parallel scheduling policies under workloads based on the results in §2. A performance comparison of these policies is provided in §4, and our concluding remarks are given in §5. Throughout this paper, we omit some of the technical details and results of our study in the interest of space, and we refer the interested reader to [11].

2 Program and Workload Behavior Models and Analysis

A general model of program and workload behavior is formulated that captures the computation and I/O characteristics of parallel applications. We derive an analysis of the model, including a generalization of Amdahl's law [1], which is then used to characterize the behavior of parallel programs based on actual measurements. Throughout the remainder of the paper, we use $\mathbb{Z}^+$ and $\mathbb{R}^+$ to denote the set of positive integer and positive real numbers, respectively.

2.1 Parallel Program Behavior

The I/O properties of many parallel programs result in execution behavior that can be naturally partitioned into disjoint intervals, each of which consists of a single burst of I/O activity (possibly with a minimal amount of computation) followed by a single burst of computation (possibly with a minimal amount of I/O). We use the term *phase* to refer to each such interval composed of an I/O burst followed by a computation burst, and we refer to a sequence of consecutive (sets of) phases that are statistically identical as an *I/O working set*. The execution behavior of a program is therefore comprised of a sequence of I/O working sets. This general model of program behavior is consistent with results from recent measurement studies (e.g., [14, 15]). As a simple example, Fig. 1 plots measurement data for a parallel implementation of the Schwinger Multichannel method for calculating low-energy electron-molecule collisions (ESCAT).

Consider such a program executed on one processor using one disk (or any minimum resource allocation required by an application). Let $N \in \mathbb{Z}^+$ denote the number of phases for the program of interest, and define $T_n \in \mathbb{R}^+$ to be the duration, or *length*, of the nth phase, $1 \leq n \leq N$.[1] The relative length of the nth phase is then given by $L_n = T_n/\mathcal{T}$,

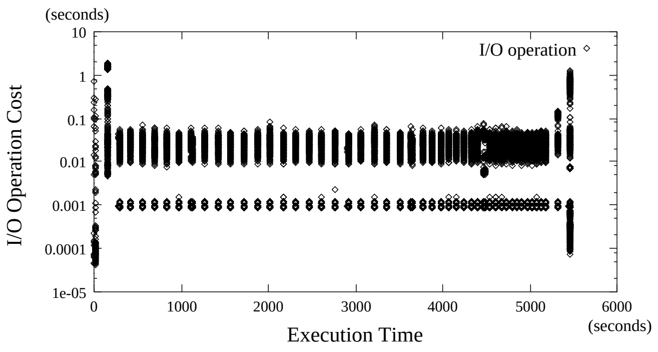


Figure 1: Measurements of program behavior for ESCAT.

where $\mathcal{T} \equiv \sum_{n=1}^{N} T_n$ is the total execution time of the program. Let T_n^{IO} and T_n^{CPU} be the length of the I/O and computation bursts of the nth phase, respectively, and thus $T_n = T_n^{\mathrm{IO}} + T_n^{\mathrm{CPU}}$. We then define $F_n = T_n^{\mathrm{IO}}/T_n$ to be the fraction of the nth phase that represents the length of the phase's I/O burst; hence, $1 - F_n$ is the fraction of the phase that represents computation. Note that during program initialization and termination, either T_i^{IO} or T_i^{CPU} can be (essentially) zero, $i \in \{1, N\}$.

A simple example of these model parameter definitions is graphically illustrated in Fig. 2, where (a) and (b) show the phase behavior of a program with respect to absolute and relative execution time, respectively. We refer to the relative execution time diagram as the *phase signature* for the program. This example consists of an initial phase composed solely of a small amount of computation (i.e., $F_1 = 0$), followed by a phase that primarily reads the problem data from disk (i.e., $F_2 \approx 0.67$). The program then repeats three phases that are computationally intensive (i.e., $F_i \approx 0.33$, $3 \leq i \leq 5$), which together comprise the first of two multi-phase I/O working sets in this example. The subsequent I/O working set is comprised of two balanced phases (i.e., $F_i \approx 0.5$, $i = 6, 7$), representing the second multi-phase working set. The final phase consists of writing the results to disk (i.e., $F_8 \approx 0.8$).

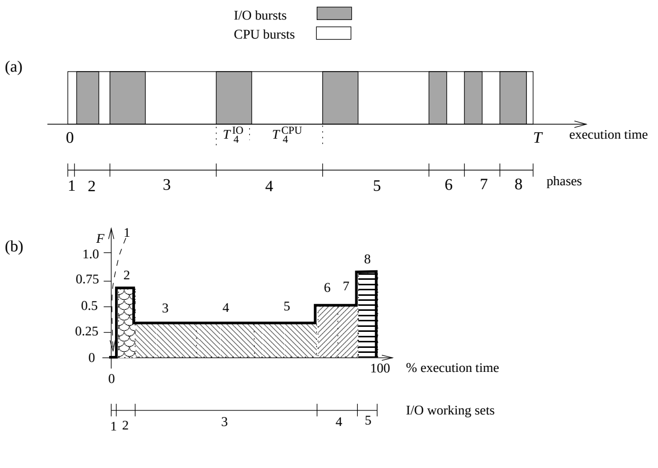


Figure 2: Example of program behavior with $N = 8$ phases: (a) absolute execution time and (b) phase signature.

We now consider a parallel system consisting of P identical processors and D identical disks. When the program of interest is executed on p processors and d disks, the computation and the I/O requirements of the program have both parallel and sequential components, as previously observed for the computational aspects of parallel programs [1]. Let $o^{\mathrm{CPU}}_{n,p,d}$ and $o^{\mathrm{IO}}_{n,p,d}$ respectively denote the duration of the sequential components for the computation and I/O bursts of the n^{th} phase when executed on p processors and d disks, $1 \le p \le P$, $1 \le d \le D$.[1] Define $s^{\mathrm{CPU}}_{n,p,d}$ and $s^{\mathrm{IO}}_{n,p,d}$ to be the respective scaling factors for the lengths of the parallel component of the computation and I/O bursts for the n^{th} phase of the program when it is executed on p processors using d disks. The execution time for each component of the n^{th} phase can then be expressed as

$$T^x_{n,p,d} \;=\; s^x_{n,p,d}\,(T^x_n \;-\; o^x_{n,1,1}) \;+\; o^x_{n,p,d}, \qquad (1)$$

where $s^x_{n,1,1} \equiv 1$, $s^x_{n,p,d} \ge 0$, $x \in \{IO, CPU\}$.[1]

Each of the above model parameters are assumed to be r.v.s with general probability distribution functions (PDF) and finite expectations. For a given program with a particular data set executed on a specific system configuration, these variables may be deterministic. When the program is executed on different data sets, however, the PDFs for these r.v.s capture the differences in execution times. Moreover, an entire class of applications, or an entire application workload, can be represented via the PDFs for the r.v.s in eq. (1).

Let $\widehat{N}$ denote the maximum number of phases in the class of applications of interest, i.e., $\widehat{N} \equiv \sup\{n : \mathrm{P}[N \ge n] > 0,\ n \in \mathbb{Z}^+\}$. The expected execution time $\mathrm{E}[\mathcal{T}_{p,d}]$ of this class of applications when executed on p processors and d disks can then be expressed as $\mathrm{E}[\mathcal{T}_{p,d}] = \sum_{n=1}^{\widehat{N}} \mathrm{E}\left[T^{\mathrm{IO}}_{n,p,d} \cdot I_{\{N\ge n\}}\right] + \mathrm{E}\left[T^{\mathrm{CPU}}_{n,p,d} \cdot I_{\{N \ge n\}}\right]$, where I_A denotes the indicator function for event A having the value 1 if A occurs and the value 0 if event A does not occur. Since the r.v. N is a stopping time, then $T^x_{n,p,d}$ and $I_{\{N\ge n\}}$ are independent for each n, and thus

$$\mathrm{E}[\mathcal{T}_{p,d}] = \sum_{n=1}^{\widehat{N}} \mathrm{E}\left[T^{\mathrm{IO}}_{n,p,d}\right] \overline{\mathcal{F}}_n + \mathrm{E}\left[T^{\mathrm{CPU}}_{n,p,d}\right] \overline{\mathcal{F}}_n \qquad (2)$$

where $\overline{\mathcal{F}}_n \equiv \mathrm{P}[N \ge n]$.

This expression can be simplified considerably, with the exact form depending upon the properties of N, $T^x_{n,p,d}$ and the elements of $T^x_{n,p,d}$ for the class of applications of interest. As a specific example, upon substituting eq. (1) into (2) with $Z^x_n \equiv T^x_n - o^x_{n,1,1}$ and the r.v.s $s^x_{n,p,d}$ and Z^x_n independent, we can simplify eq. (2) to

$$\mathrm{E}[\mathcal{T}_{p,d}] = \sum_{n=1}^{\widehat{N}} \overline{\mathcal{F}}_n \,\left(\mathrm{E}\left[s^{\mathrm{IO}}_{n,p,d}\right]\mathrm{E}\left[Z^{\mathrm{IO}}_n\right] + \mathrm{E}\left[o^{\mathrm{IO}}_{n,p,d}\right] + \mathrm{E}\left[s^{\mathrm{CPU}}_{n,p,d}\right]\mathrm{E}\left[Z^{\mathrm{CPU}}_n\right] + \mathrm{E}\left[o^{\mathrm{CPU}}_{n,p,d}\right]\right). \qquad (3)$$

This case is chosen here because its assumptions hold for, and thus it is used to obtain, the results presented in §2.5. The simplified forms of eq. (2) for all other cases can be found in [11], together with the details on all derivations.

2.2 Performance Metrics

An important and standard measure to characterize the behavior of a parallel program is *speedup*, which in the present context is defined as the ratio of the execution time on a single processor and disk to the execution time on p processors and d disks. In the stochastic setting, speedup can be expressed in a number of different ways. One particular definition of interest here is given by

$$S_{p,d} \;=\; \frac{\mathrm{E}[\,\mathcal{T}_{1,1}\,]}{\mathrm{E}[\,\mathcal{T}_{p,d}\,]}, \qquad (4)$$

which describes the *speedup surface* in the space $(S_{p,d}, p, d)$. Note that this expression for speedup is consistent with that used in [6], where speedup is defined to be the ratio of the expected response time in a queueing system with one processor to the expected response time in this system with multiple processors. We also note, however, that alternative expressions for speedup are considered in [11].

The *efficiency* of a program is typically defined as the speedup per processor allocated to the application. Since the resources allocated in the present context include the disks used by the application, we express program efficiency as $E_{p,d} = S_{p,d}/g(p,d) = \mathrm{E}[\mathcal{T}_{1,1}]/(g(p,d)\mathrm{E}[\mathcal{T}_{p,d}])$, where $g(p,d)$ is a general function that can be used to scale the speedup according to any desired weightings of the two resource allocations. Note that the traditional upper bound of unity for efficiency holds when $d = 1$, but in general the upper bound depends upon the specific form of the function $g(p,d)$.

The speedup can also be scaled by a general cost function of the resources allocated, $C(p,d)$, which we define here to be $C(p,d) = g(p,d)/S_{p,d}$. This yields a particular measure called program *efficacy* as follows: $\eta_{p,d} = S^2_{p,d}/g(p,d) = \mathrm{E}[\mathcal{T}_{1,1}]^2/(g(p,d)\mathrm{E}[\mathcal{T}_{p,d}]^2)$.

2.3 Generalized Amdahl's Law

Our model and analysis of §2.1 and §2.2 can be used to derive several fundamental properties of the behavior of parallel programs. We now present one such derivation that extends Amdahl's law to three-dimensional space to include processors and disks.

Upon substituting (2) into eq. (4) and multiplying both the numerator and denominator by $\mathrm{E}[\mathcal{T}_{1,1}]^{-1}$, we obtain an expression that under the assumptions of eq. (3) leads to

$$S_{p,d} = \{\ \mathrm{E}[\mathcal{T}_{1,1}]^{-1} \sum_{n=1}^{\widehat{N}} (\mathrm{E}\left[s^{\mathrm{IO}}_{n,p,d}\right]\mathrm{E}\left[Z^{\mathrm{IO}}_n\right] + \mathrm{E}\left[o^{\mathrm{IO}}_{n,p,d}\right] + \mathrm{E}\left[s^{\mathrm{CPU}}_{n,p,d}\right]\mathrm{E}\left[Z^{\mathrm{CPU}}_n\right] + \mathrm{E}\left[o^{\mathrm{CPU}}_{n,p,d}\right])\,\overline{\mathcal{F}}_n\ \}^{-1}. \qquad (5)$$

This expression, together with the corresponding eqs. in [11] for different assumptions, can be viewed as general forms of Amdahl's law, with more simple forms obtained by making additional assumptions.

Define $f^x_{\mathrm{par}} \equiv \sum_{n=1}^{\widehat{N}} \mathrm{E}[Z^x_n]\,\overline{\mathcal{F}}_n/\mathrm{E}[\mathcal{T}_{1,1}]$ and $f^x_{\mathrm{seq}} \equiv \sum_{n=1}^{\widehat{N}} \mathrm{E}\left[o^x_{n,p,d}\right]\overline{\mathcal{F}}_n/\mathrm{E}[\mathcal{T}_{1,1}]$ to be the cumulative parallel and sequential fractions of the x-component in a parallel program, respectively. Under the assumption that the r.v.s $s^{\mathrm{IO}}_{n,p,d}$ and $s^{\mathrm{CPU}}_{n,p,d}$ are identical for all phases n, i.e., $s^x_{n,p,d} = h^{-1}_x(p,d)$, we can express eq. (5) as

$$S_{p,d} \;=\; \left\{\ f^{\mathrm{IO}}_{\mathrm{par}} \cdot \mathrm{E}\left[h^{-1}_{\mathrm{IO}}(p,d)\right] \;+\; f^{\mathrm{IO}}_{\mathrm{seq}} + f^{\mathrm{CPU}}_{\mathrm{par}} \cdot \mathrm{E}\left[h^{-1}_{\mathrm{CPU}}(p,d)\right] \;+\; f^{\mathrm{CPU}}_{\mathrm{seq}}\ \right\}^{-1}, \qquad (6)$$

where $h^{-1}_x(p,d)$ are general functions that can be used to scale the speedup according to any desired weightings of the

[1]All subsequent uses of the indices n, p, d and x are intended to be under the constraints $1 \le n \le N$, $1 \le p \le P$, $1 \le d \le D$ and $x \in \{IO, CPU\}$, respectively, unless noted otherwise.

two resource allocations. An even simpler form can be obtained by setting $\mathrm{E}\left[h_{\mathrm{IO}}^{-1}(p,d)\right] = d^{-1}$ and $\mathrm{E}\left[h_{\mathrm{CPU}}^{-1}(p,d)\right] = p^{-1}$ in (6), which yields

$$S_{p,d} \;=\; \left\{\, f_{\mathrm{seq}}^{\mathrm{IO}} + f_{\mathrm{seq}}^{\mathrm{CPU}} + f_{\mathrm{par}}^{\mathrm{IO}}/d + f_{\mathrm{par}}^{\mathrm{CPU}}/p \,\right\}^{-1}. \qquad (7)$$

Note that eq. (7) reduces to the original version of Amdahl's law upon ignoring the I/O component of the parallel program and observing that $f_{\mathrm{seq}}^{\mathrm{CPU}} + f_{\mathrm{par}}^{\mathrm{CPU}} = 1$.

2.4 Workload Model and Analysis

A general model of parallel workloads can be formulated as a mixture of applications, each of which is represented by the model and analysis of §2.1–2.3. Although we omit the details here, our analysis leads quite naturally to an expression of the form given in (2) where the PDFs for the variables on the RHS capture the details of the parallel workload as a whole in addition to individual (classes of) applications. Given this form, one can follow the derivation in the previous sections to obtain a series of expressions analogous to those in (3) through (7) to characterize a general parallel workload.

2.5 Program Behavior Results

We now apply our models and analysis to characterize the behavior of real parallel applications, demonstrating that our methodology is an extremely accurate and powerful abstraction for application scalability as a function of its resource requirements. A representative sample is provided here.

Our experiments were conducted on a 512-node Intel Paragon XP/S with 64 4-GB Seagate disks, using Pablo [10] to collect performance data. Each application was executed in single-user mode on the assigned processor and disk partitions, the sizes of which were varied from 1 to 64. Application data were distributed on the disk partition according to the Intel Parallel File System strategy using the default striping unit (i.e., 64 KB) starting from a randomly selected disk, so as to keep the load balanced across the disks. Alternative striping parameters were not considered because our interest is in analyzing application resource requirements rather than investigating parallel file system performance.

The applications used in our experiments consisted of those in the Scalable Input/Output Initiative (SIO) suite that have non-negligible resource requirements. Each application consists of a number of programs, or *stages*, executed in a pipeline fashion. The measurements for the individual stages of each application were used to parameterize the program behavior model described in §2.1–2.3. We first note that comparisons between the estimates from the eqs. in §2.2 and the corresponding measurement data for the entire application show that the results from our model are in excellent agreement with the measurements. We further note that our model can significantly simplify the measurement process by allowing one to measure each unique phase in isolation. This can also lead to more accurate performance data by reducing the likelihood and impact of external events on the measured times for long running applications.

Our program behavior model can also be used to accurately estimate the value of performance metrics for the cases where measurements are not available. In particular, we used our model parameterized by the measurements for smaller values of p and d to extrapolate the value of performance metrics for larger values of p and d. We then took these model predictions for the largest p and d values for which we also had total execution time measurement data, and compared them with the corresponding results obtained directly by extrapolating from the overall execution time measurements for smaller values of p and d. Our results demonstrate that the model estimates for these cases are essentially identical to the measurement data, and they are significantly more accurate than those obtained from the overall execution time measurements. The estimates based on our model can lead to considerably more accurate predictions than those obtained purely from total execution times because the model captures the computation and I/O characteristics of the parallel applications in detail.

We next consider the three-dimensional speedup surfaces obtained from our modeling analysis parameterized by the measurement data. In particular, Fig. 3 illustrates the speedup function from our models for the QCRD (quantum chemical reaction dynamics) and MESSKIT (electronic structure calculations via the Hartree-Fock algorithm) applications with respect to both the processor and disk partition sizes. These results clearly illustrate that speedup does not necessarily increase as the number of allocated disks and processors increase. In fact, the quantitative computation and I/O requirements of the application, and their interactions, are among the important factors determining its ideal number of processors and disks.

To examine more carefully the relative computation and I/O requirements of the applications, we focus on the phase behavior of QCRD's second stage and MESSKIT's SCF stage. The phase behavior and phase signature for each program is obtained via the analysis of §2.1. QCRD's phase signature is illustrated in Fig. 4 and SCF's phase signature is illustrated in Fig. 5. As indicated by both figures, the I/O activity can be characterized as repetitive and bursty, with 12 distinct phases for QCRD and 5 distinct phases for SCF. Since for both programs the overall signature is roughly uniform, each application is characterized by a single I/O working set. We ignore the negligible sequential I/O and computation activity at the beginning and at the end of each program. Once the I/O working set is identified, it is trivial to apply our methodology and derive the expected application execution time.

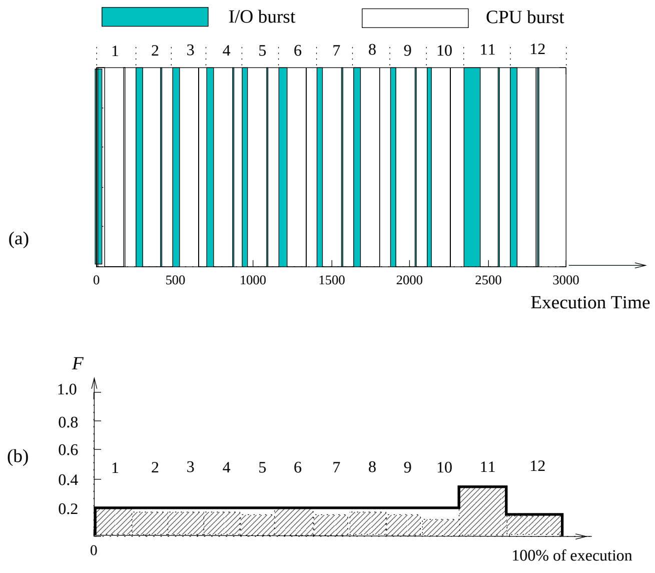
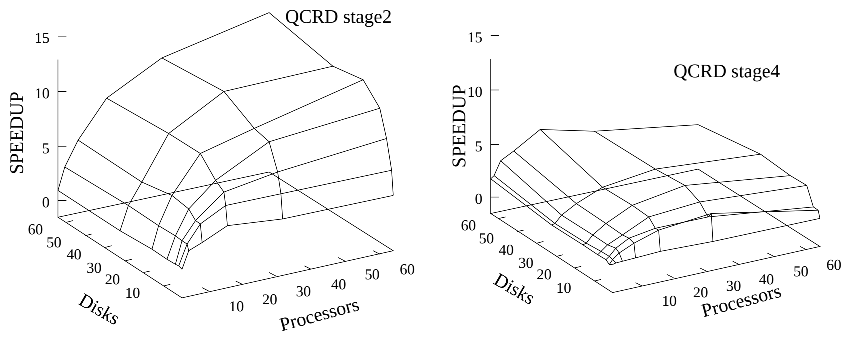


Figure 4: Phase behavior and signature of QCRD's stage 2.



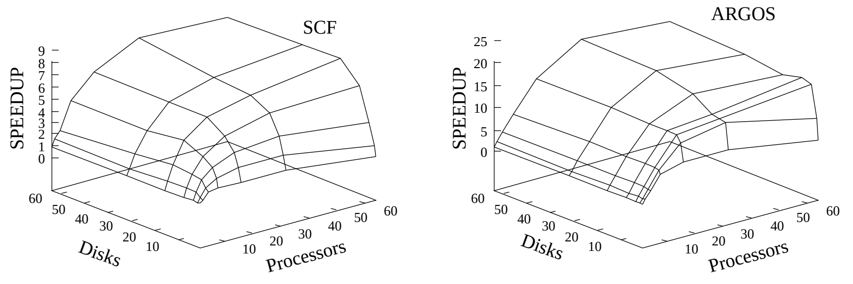


Figure 3: Speedup surface of QCRD (stages two and four) and MESSKIT (stages ARGOS and SCF).

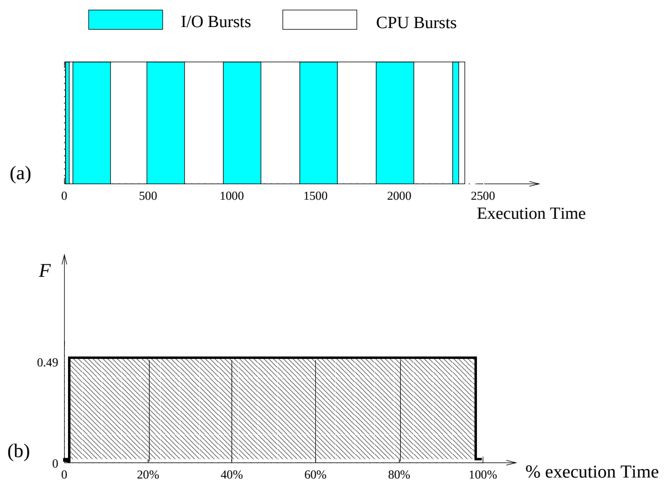


Figure 5: Phase behavior and signature of MESSKIT's SCF.

3 Scheduling Models and Analysis

Exploiting the analysis of §2 to characterize parallel application and workload behavior, in this section we develop a formal framework to investigate the coordinated allocation of processor and disk resources in parallel systems. First, we present our parallel system model, and then we derive a general analysis of the sojourn time through a disk partition when multiple processor partitions share these disks based on decomposition and a generalization of standard flow-equivalent service centers [4]. Some of the scheduling policies considered in our study, and specific aspects of their respective scheduling models, are briefly described in turn.

3.1 Parallel System Model

We consider a parallel system consisting of P identical processors and D identical disks, both of which are allocated according to a particular scheduling policy. An FCFS infinite-capacity queue is used to hold all jobs that are waiting to be allocated resources. Let M_P and M_D denote the minimum number of processors and disks allocated to any parallel job, respectively, and therefore the maximum number of processor and disk partitions are respectively given by $N_P = \lfloor P/M_P \rfloor$ and $N_D = \lfloor D/M_D \rfloor$; hence, $p \geq M_P$ and $d \geq M_D$ throughout.

The interarrival times of parallel jobs are modeled as i.i.d. $\mathbb{R}^+$-valued r.v.s having PDF $\mathbf{A}(\cdot)$, whereas the parallel job service times depend upon the number of processors and disks allocated to the job. Following our model of §2 for the entire workload, the per-phase service times are modeled as independent sequences of i.i.d. $\mathbb{R}^+$-valued r.v.s having PDFs $\mathbf{B}^{\text{IO}}_{n,p,d}(\cdot)$ and $\mathbf{B}^{\text{CPU}}_{n,p,d}(\cdot)$. The PDFs for all parameters of our scheduling models are assumed to be of PH-type. In particular, $\mathbf{A} = \text{PH}(\underline{\alpha}, \mathcal{S}_A)$ of order m_A with mean $\lambda^{-1} = -\underline{\alpha}\mathcal{S}_A^{-1}\mathbf{e} < \infty$, and $\mathbf{B}^x_{n,p,d} = \text{PH}(\underline{\beta}^x_{n,p,d}, \mathcal{S}^x_{B,n,p,d})$ of order $m^x_{B,n,p,d}$ with mean $\mu^{x\ -1}_{n,p,d} = -\underline{\beta}^x_{n,p,d}\mathcal{S}^{x\ -1}_{B,n,p,d}\mathbf{e} < \infty$, where $\mathbf{e}$ denotes the column vector of all ones. It follows directly from the convolution properties of PH-type PDFs that the total execution time of a job when exclusively executed on a processor partition of size p and a disk partition of size d is an order $\sum_{n=1}^{N} m^{\text{IO}}_{B,n,p,d} + m^{\text{CPU}}_{B,n,p,d}$ PH-type PDF [11].

The use of PH-type PDFs is of theoretical importance in that we exploit their mathematical properties to derive tractable solutions to aspects of our general analysis while capturing the fundamental elements of the parallel scheduling systems of interest. It is also of practical importance in that a number of algorithms have been developed for fitting PH-type PDFs to empirical data.

3.2 Decomposition of I/O Center

The parallel system is modeled as an open queueing network with a multi-server processor center and a multi-server disk center. However, we can significantly simplify our analysis of the stochastic processes for the space-sharing scheduling models by using a single composite center to represent the processor and disk partitions. In particular, if the processor and disk partitions are not shared by multiple jobs, an equivalent queueing system is obtained by using the convolution of the per-phase computation and I/O PDFs $\mathbf{B}^{\text{CPU}}_{n,p,d}(\cdot)$ and $\mathbf{B}^{\text{IO}}_{n,p,d}(\cdot)$ for the single-center service time PDF. Otherwise, when there is contention for a disk partition, we use decomposition to analyze the corresponding processor partitions and disk partition in isolation and obtain the corresponding sojourn-time PDF(s) for a generalized flow-equivalent center. The flow-equivalent PDF(s) for the time spent in each I/O phase is then used together with the per-phase computation PDFs to characterize the total execution time at the composite center.

We now derive the details of our approach for the case where the phases basically have the same PDF. This simplifies the presentation of our approach, and this assumption holds for the applications considered in our numerical experiments presented in §4. Note, however, that our general approach can address the case where different phases have different PDFs by using a multiple-class queueing network in the analysis of the aggregate network representing the processor partitions and disk partition in isolation, which is viable in practice due to our efficient solution.

Consider a closed two-center queueing network with a single delay center, representing the set of $Y \geq 2$ processor partitions sharing the disk partition, and a single queueing center, representing the disk partition. The service times at the computation center follow an order m_P PH-type PDF $\mathcal{B}_P = \text{PH}(\underline{\gamma}, \mathcal{S}^P_{\mathcal{B}})$ with mean $Z_P = -\underline{\gamma}\mathcal{S}^{P\,-1}_{\mathcal{B}}\mathbf{e}$, which is constructed from the PDFs $\mathbf{B}^{\text{CPU}}_{n,p,d}(\cdot)$ for the processor partitions being considered. Similarly, the service times at the disk center follow an order m_D PH-type PDF $\mathcal{B}_D = \text{PH}(\underline{\delta}, \mathcal{S}^D_{\mathcal{B}})$ with mean $Z_D = -\underline{\delta}\mathcal{S}^{D\,-1}_{\mathcal{B}}\mathbf{e}$, which is constructed from the PDFs $\mathbf{B}^{\text{IO}}_{n,p,d}(\cdot)$ for the disk partition of interest.

We represent this queueing network as a CTMC $\{X(t)\,;\, t \geq 0\}$ defined on a finite state space Ω, whose elements are given by $(i, \underline{j}^D, \underline{j}^P)$ where $\underline{j}^z \equiv (j^z_1, j^z_2, \ldots, j^z_{m_z})$, $z \in \{D, P\}$, $j^D_\ell \geq 0$, $\sum_{\ell=1}^{m_D} j^D_\ell = i$, $j^P_\ell \geq 0$, $\sum_{\ell=1}^{m_P} j^P_\ell = Y - i$. The state vector variable $i \in \{0, \ldots, Y\}$ denotes the number of jobs at the disk center, $j^D_\ell \in \{0, \ldots, i\}$ denotes the number of jobs at the disk center whose service time process is in stage ℓ, and $j^P_\ell \in \{0, \ldots, Y - i\}$ denotes the number of jobs at the computation center whose service time process is in stage ℓ. For convenience, we also partition the state space Ω into $Y + 1$ disjoint sets $\Omega_i \equiv \left\{(i, \underline{j}^D, \underline{j}^P) \mid (i, \underline{j}^D, \underline{j}^P) \in \Omega\right\}$, $0 \leq i \leq Y$.

Let $y_{i,w} \in \Omega_i$, $1 \leq w \leq D_i \equiv |\Omega_i|$, $0 \leq i \leq Y$, be a lexicographical ordering of the elements of Ω_i. We then define $\boldsymbol{\pi} \equiv (\boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_Y)$, $\boldsymbol{\pi}_i \equiv (\pi(y_{i,1}), \pi(y_{i,2}), \ldots, \pi(y_{i,D_i}))$ and $\pi(y_{i,w}) \equiv \lim_{t\to\infty} \text{P}\left[X(t) = y_{i,w}\right]$, $y_{i,w} \in \Omega_i$, $1 \leq w \leq D_i$, $0 \leq i \leq Y$. Assuming the stochastic process $\{X(t), t \geq 0\}$ to be irreducible and positive recurrent, the invariant probability vector is uniquely determined by solving the global balance eqs. $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ together with the normalizing constraint $\boldsymbol{\pi}\mathbf{e} = 1$, where $\mathbf{Q}$ is the infinitesimal generator matrix for the process.

The generator $\mathbf{Q}$, arranged in the same order as the elements of the stationary probability vector $\boldsymbol{\pi}$, is given by

$$\mathbf{Q} = \begin{bmatrix} A_{0,1} & A_{0,0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ A_{1,2} & A_{1,1} & A_{1,0} & \cdots & & \\ \mathbf{0} & A_{2,2} & A_{2,1} & \cdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & A_{Y-2,0} & \mathbf{0} \\ & & & & A_{Y-1,1} & A_{Y-1,0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & A_{Y,2} & A_{Y,1} \end{bmatrix}, \quad (8)$$

where $A_{i,2}$, $A_{i,1}$ and $A_{i,0}$ are finite matrices of dimensions $D_i \times D_{i-1}$, $D_i \times D_i$ and $D_i \times D_{i+1}$, respectively, $0 \leq i \leq Y$. A simple yet general method for constructing these matrices for all instances of our model can be found in [11].

The invariant vector $\boldsymbol{\pi}$ then satisfies the eqs.

$$\boldsymbol{\pi}_0 A_{0,1} + \boldsymbol{\pi}_1 A_{1,2} = \mathbf{0}, \quad (9)$$

$$\boldsymbol{\pi}_{k-1} A_{k-1,0} + \boldsymbol{\pi}_k A_{k,1} + \boldsymbol{\pi}_{k+1} A_{k+1,2} = \mathbf{0}, \quad (10)$$

$$\boldsymbol{\pi}_{Y-1} A_{Y-1,0} + \boldsymbol{\pi}_Y A_{Y,1} = \mathbf{0}, \quad (11)$$

and $\boldsymbol{\pi}\mathbf{e} = 1$, $1 \leq k \leq Y - 1$. Define $R(Y) \equiv A_{Y,1}$, $R(k) \equiv A_{k,1} - A_{k,2}R(k-1)^{-1}A_{k-1,0}$, $0 \leq k \leq Y - 1$, and $R(-1) \equiv \mathbf{0}$. Substituting this into eqs. (9)–(11), we obtain

$$\boldsymbol{\pi}_k = -\boldsymbol{\pi}_{k+1} A_{k+1,2} R(k)^{-1}, \quad 0 \leq k \leq Y - 1, \quad (12)$$

$$\boldsymbol{\pi}_Y = -\boldsymbol{\pi}_{Y-1} A_{Y-1,0} R(Y)^{-1}, \quad (13)$$

where $R(k)$ has dimension $D_k \times D_k$ and is non-singular. The vector $\boldsymbol{\pi}_{Y-1}$ can then be determined up to a multiplicative constant by solving $\boldsymbol{\pi}_{Y-1}\phi = \mathbf{0}$ and $\boldsymbol{\pi}_{Y-1}\mathbf{e} = 1$, where $\phi = A_{Y-1,1} - A_{Y-1,2}R(Y-2)^{-1}A_{Y-2,0} - A_{Y-1,0}R(Y)^{-1}A_{Y,2}$, for $Y \geq 2$ (note that $\phi = A_{1,1} - A_{1,2}R(0)^{-1}A_{0,0}$ when $Y = 1$). Upon substitution into eqs. (12) and (13), we obtain the remaining components $\boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_{Y-2}, \boldsymbol{\pi}_Y$ up to that same constant, from which the vector $\boldsymbol{\pi}$ is uniquely determined by the normalizing eq. $\boldsymbol{\pi}\mathbf{e} = 1$. Note that this approach for calculating $\boldsymbol{\pi}$ significantly reduces the computational complexity over that of numerically solving the global balance and normalizing eqs. directly, and thus makes it possible to solve large systems that are otherwise prohibitively expensive.

Given the stationary vector $\boldsymbol{\pi}$, the k^{th} moment of the number of jobs at the two centers are given by

$$\text{E}\left[U_P^k\right] = \sum_{i=0}^{Y-1} (Y - i)^k \boldsymbol{\pi}_i \mathbf{e}, \quad \text{E}\left[U_D^k\right] = \sum_{i=1}^{Y} i^k \boldsymbol{\pi}_i \mathbf{e}, \quad (14)$$

$Y \geq 2$. From Little's law [4], the throughput for the aggregate network can be expressed as $X_{\text{A}} = \text{E}\left[U_P\right]/Z_P$, and then we can obtain the mean sojourn time through the disk center as $\text{E}\left[Z'_D\right] = \text{E}\left[U_D\right]/X_{\text{A}}$. Higher moments of the sojourn time Z'_D can be obtained from (14), $\mathcal{B}_P$, $\boldsymbol{\pi}$ and generalizations of Little's law (see [11] for the technical details and references). We then construct an order m'_D PH-type PDF $\mathcal{B}'_D = \text{PH}(\underline{\delta}', \mathcal{S}^{D'}_{\mathcal{B}})$ with mean $\text{E}\left[Z'_D\right] = -\underline{\delta}'\mathcal{S}^{D'\,-1}_{\mathcal{B}}\mathbf{e}$, to match as many moments of Z'_D as desired. This PDF(s) for the per-phase sojourn time through the disk partition is then used together with the per-phase computation PDF(s) $\mathcal{B}_P = \text{PH}(\underline{\gamma}, \mathcal{S}^P_{\mathcal{B}})$ and convolution to characterize the total execution time at the composite center in our analytic space-sharing models.

3.3 Static Space-Sharing

The static space-sharing strategy $\text{SP}(K_P, K_D)$ divides the P processors into K_P partitions of size $\lfloor P/K_P \rfloor$ or $\lceil P/K_P \rceil$

such that $\lfloor P/K_P \rfloor \geq M_P$, and similarly divides the D disks into K_D partitions of size $\lfloor D/K_D \rfloor$ or $\lceil D/K_D \rceil$ such that $\lfloor D/K_D \rfloor \geq M_D$. Processor partitions are used exclusively, whereas disk partitions may be shared. Each job arrival is allocated one such pair of partitions if one is available, but otherwise the job waits in the FCFS queue until a partition pair becomes free. Every parallel job is executed to completion without interruption and all of the resources in the processor partition are reserved by the application throughout this duration. Upon job departure, the available partition pair is allocated to the job at the head of the system queue.

3.4 Dynamic Space-Sharing

The dynamic space-sharing strategy $\mathrm{DP}(K_D)$ attempts to equally allocate the P processors among the parallel jobs in the system, whereas the disks are divided into K_D partitions as in $\mathrm{SP}(K_P, K_D)$. In particular, upon the arrival of a job when the system is executing i jobs, $0 \leq i < N_P$, the P processors are repartitioned among the $i+1$ jobs such that each job is allocated a processor partition either of size $\lfloor P/(i+1) \rfloor$ or of size $\lceil P/(i+1) \rceil$. A job arrival that finds $i \geq N_P$ jobs in the system is placed in the FCFS queue to wait until a processor partition becomes available. When one of the i jobs in the system departs, $2 \leq i \leq N_P$, the system reconfigures the processor allocations among the remaining $i-1$ jobs such that there are only partitions of sizes $\lfloor P/(i-1) \rfloor$ and $\lceil P/(i-1) \rceil$. A departure when the system contains more than N_P jobs causes the job at the head of the queue to be allocated the available processor partition, and no reconfiguration is performed. The times required to repartition the processors among the i jobs to be executed are modeled as i.i.d. r.v.s having an order $m_{C,i}$ PH-type PDF $\mathbf{C}_i = \mathrm{PH}(\underline{\zeta}_i, \mathcal{S}_{C,i})$ with mean $\theta_i^{-1} = -\underline{\zeta}_i \mathcal{S}_{C,i}^{-1} \mathbf{e} < \infty$, $1 \leq i \leq N_P$.

3.5 Static Space-Sharing with Time-Sharing

The static space-sharing with time-sharing strategy $\mathrm{TS}(K_P, K_D)$ divides the processors and disks in scheduling pairs as in $\mathrm{SP}(K_P, K_D)$. The key feature of this policy is that each processor-disk pair may be shared up to a maximum multiprogramming level of $\mathcal{M}$ jobs. The policy takes advantage of the bursty CPU and I/O behavior of the jobs by interleaving $\mathcal{M}$ jobs on each processor and disk partition pair. In this manner, it attempts to keep both resources simultaneously busy. Essentially, the computation bursts of some of the jobs assigned to a partition are overlapped with the I/O bursts of the partition's other jobs. The extent to which such an overlap is advantageous depends upon the phase characteristics of each individual job scheduled on the partition pair.

A processor-disk partition pair is available as long as there is a processor partition whose current multiprogramming level is less than $\mathcal{M}$. When a job arrives at the system one of the following four cases is possible. If there is a processor-disk pair where both partitions are idle, then the newly arrived job is assigned to this pair. Otherwise, the policy selects a pair where at least one of the two partitions is idle. If there is a scheduling slot where both partitions are already busy with less than $\mathcal{M}$ jobs each, then the job is scheduled onto it. Finally, if all partition pairs are assigned $\mathcal{M}$ jobs each, then there are no available scheduling slots and the job waits in the queue. The maximum number of jobs simultaneously executing in the system is $\mathcal{M} \times \min\{K_P, K_D\}$.

4 Scheduling Results

In this section we present a representative sample of the results from our scheduling analysis based on §3 under a workload based on the models of §2 parameterized by measurements of seven programs from the SIO suite. A parallel system with $P = 128$, $D = 32$, $M_P = 8$, $M_D = 1$, $K_P \in \{1, 2, 4, 8, 16\}$ and $K_D \in \{1, 2, 4, 8, 16, 32\}$ is considered. The job interarrival and service times have coefficients of variation that cover the range of statistics reported in previous measurement studies of workloads at supercomputing centers. The performance measures for our space-sharing models are obtained using the exact analysis (with state-dependent PDFs) derived in [17] together with the analysis of §3.2, whereas the scheduling systems with time-sharing are analyzed via simulation. Throughout, we use ρ to denote the traffic intensity of the parallel system, $0 < \rho < 1$.

4.1 Space-Sharing Response Time

We first consider the response time measures of the system under space-sharing strategies. Previous research on parallel space-sharing policies, when only the scheduling of processors is considered and I/O behavior is essentially ignored, has basically shown that the optimal static partitioning scheme at light system loads consists of fewer (thus larger) partitions, and that the optimal static partitioning scheme consists of increasingly more (hence smaller) partitions as the traffic intensity rises, with the effects of memory-constrained environments being an exception [9]. This results in a family of response time curves for the various possible partition sizes whose relative ordering, with respect to providing the best performance, typically follows the natural order on the number of partitions, from 1 (i.e., the entire system) to N_P (i.e., the maximum number of processor partitions). Furthermore, dynamic partitioning policies have often been shown to provide a lower mean response time than all of the static space-sharing policies across the entire range of system loads.

We now exploit the space-sharing models in §3, which explicitly consider a workload with non-negligible I/O requirements, to examine the performance of coordinated processor and disk allocation policies and compare the corresponding trends with previous scheduling results. The mean job response times for the space-sharing systems with different numbers of processor and disk partitions are plotted in Fig. 6. We observe that when K_D is greater than or equal to the entire set of K_P values considered, then our results are basically consistent with previous scheduling results in the research literative. Specifically, for $K_D \geq 16$, a single processor partition provides the best performance among the static policies at very light loads. When ρ is quite small, job response times are essentially dominated by their processing requirements given the number of allocated processors, and hence the benefits of maximizing the parallel execution of jobs outweigh its associated costs. As ρ increases from near 0 towards 1, however, $K_P = 2$ provides the best static partitioning performance, followed by $K_P = 4$ and then $K_P = 8$. This is because queueing delays have an increasingly larger relative effect on job response times as the traffic intensity rises, which together with the non-linear speedups of the applications comprising the parallel workload tends to cause the optimal static partition size to decrease with increasing values of ρ.

The only exception to these trends is the case of $K_P = 16$, which provides poorer mean response times than the $\mathrm{SP}(8, 16)$ system across all traffic intensities. This is due to

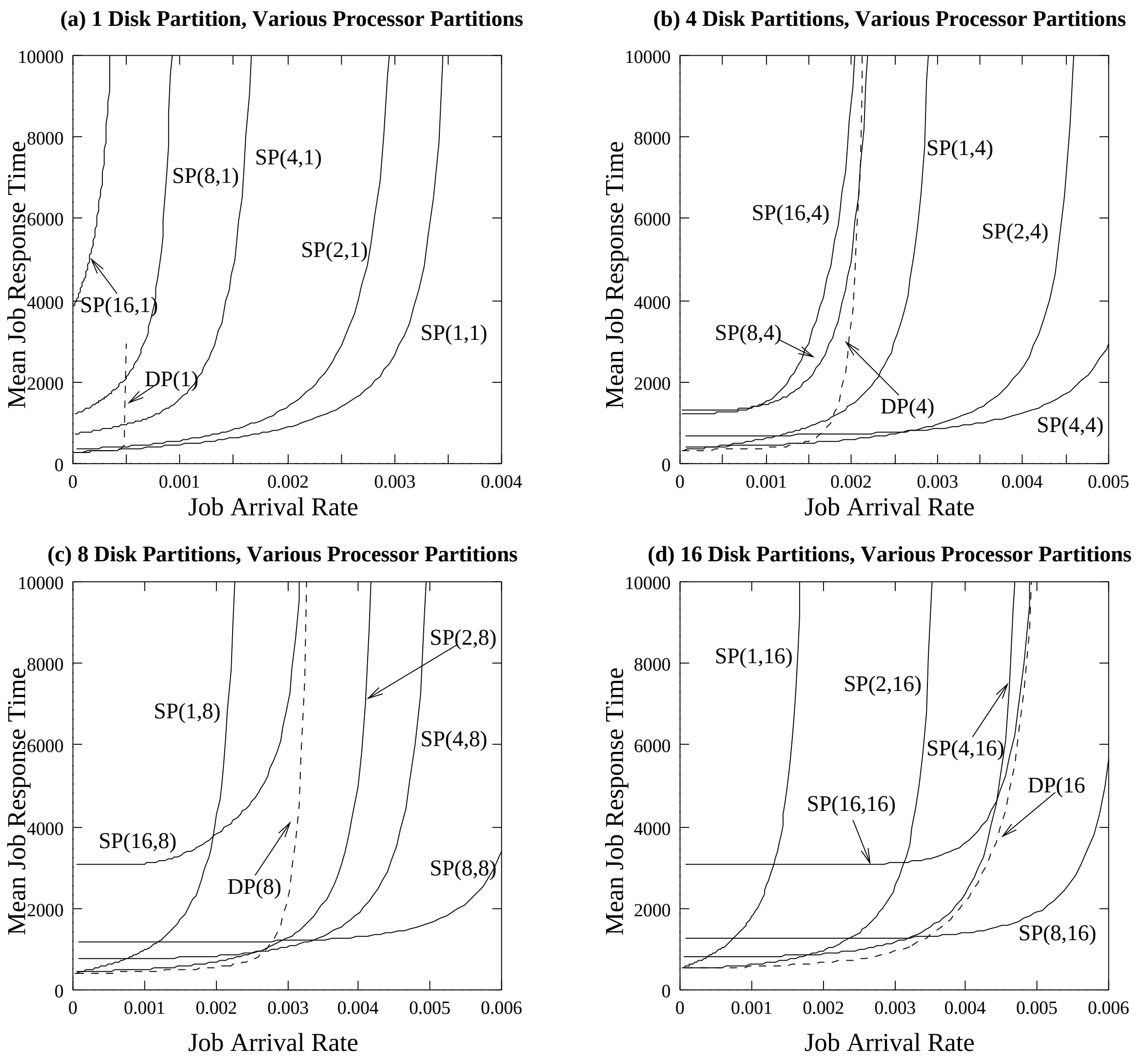


Figure 6: Mean response times under $\mathrm{SP}(K_P, K_D)$ and $\mathrm{DP}(K_D)$ for $K_P \in \{1, 2, 4, 8, 16\}$ and $K_D \in \{1, 4, 8, 16\}$.

a particular artifact of one of the parallel applications in our workload in which the execution time as a function of the number of allocated processors and disks is discontinuous. Note that in each of the above cases there is no contention for the disk resources since each processor partition has its own private disk partition. The above performance trends and observations tend to also hold for $K_D = 8$, although with changes in the range of ρ values over which a particular value of K_P is best.

On the other hand, as the value of K_D is reduced further, the performance trends previously observed in the research literature for static partitioning appear to be no longer valid. In particular, when there is a single partition consisting of all the disks, we observe that there is a strict ordering among the static space-sharing strategies for all traffic intensities, where $\mathrm{SP}(1, 1)$ provides the best performance, followed in order by $\mathrm{SP}(2, 1)$, $\mathrm{SP}(4, 1)$, $\mathrm{SP}(8, 1)$ and $\mathrm{SP}(16, 1)$, $0 < \rho < 1$. A primary cause of these performance trends is contention among the K_P processor partitions for the single disk partition, which tends to have a dominating effect over the queueing delays and non-linear speedups mentioned above. The quantitative results from our analysis of §3.2 clearly demonstrate the performance impact of these effects. Of course, this contention for the disk resources declines as the number of processor partitions decreases. Similar observations can be made with respect to the mean response times for $K_D = 2$ and $K_D = 4$, with the contention for the disk partitions decreasing as the value of K_D increases. It is important to note that the results presented in this section are based on the use of a processor sharing discipline at the disk partitions (although our analysis in §3 is not limited to this assumption and in fact supports general scheduling at the disk partitions), and thus our model at worst underestimates the contention for the disk resources.

Turning to the results for dynamic partitioning, i.e., the dashed curves in Fig. 6, we observe that $\mathrm{DP}(K_D)$ provides the best performance among all space-sharing strategies for small to relatively moderate values of ρ depending upon the value of K_D. By adjusting scheduling decisions in response to workload changes, the $\mathrm{DP}(K_D)$ system provides the most efficient utilization of the processors among the various space-sharing policies when the traffic intensity and disk contention are sufficiently low. On the other hand, as shown in [17, 16], the parallel system under dynamic partitioning converges toward the corresponding static partitioning system with $K_P = N_P$ in the limit as $\rho \to 1$. Hence, while DP(K_D) typically provides the best performance at relatively light loads, it eventually starts to suffer from the disk contention problems described above and thus tends to saturate together with the corresponding SP(N_P, K_D) system. It is important to note that the dynamic partitioning results in Fig. 6 are based on the assumption of no repartitioning overhead, i.e., $\theta_i^{-1} = 0$, and thus our results are at worst biased in favor of dynamic space-sharing.

4.2 Time Sharing Response Time

Our analysis of parallel program and workload behavior in §2 clearly identifies an important characteristic of I/O-intensive parallel applications, namely the repeated alternation between computation bursts and I/O bursts. This together with the performance trends from the above scheduling results suggest that the overlapping of the I/O demands of some jobs with the computation demands of other jobs may offer a potential improvement in performance, even with limited multiprogramming levels. We now exploit the space-sharing with time-sharing models in §3 to examine these performance issues in more detail.

The mean job response times for the TS(K_P, K_D) systems with $\mathcal{M} = 2$ and different numbers of processor and disk partitions are plotted in Fig. 7. We first observe that the overlapping of computation and I/O bursts from different jobs results in the most efficient utilization of the resources among the scheduling policies considered in our study. This also yields a significant increase in the traffic intensities accommodated by the time-sharing counterpart of the scheduling policies examined in the previous section. Moreover, the performance benefits of this effective resource utilization outweigh the impact of contention among the K_P processor partitions for the K_D disk partitions, which had a considerable effect on the performance of space-sharing policies. Note further that this contention can be even worse in TS(K_P, K_D) than in the space-sharing systems when $K_P \geq K_D$, as up to $\mathcal{M} \cdot K_P / K_D$ jobs share each disk partition.

The multiprogramming level that provides the best performance depends in part upon the ratio of the per-phase computation and I/O requirements. In the ideal case, the scheduling strategy will be able to interleave the execution of applications such that this ratio is maintained very close to 1, thus achieving a total overlapping of computation and I/O. The problem of coordinated allocation of processor and disk resources consists of a complex tradeoff that depends upon a number of key factors, including the speedup surfaces of the application workload as a function of the number of allocated resources, the system load, and the balance of the computation and I/O requirements present in the parallel workload.

5 Conclusions

In this paper we systematically investigated performance issues involved in the coordinated allocation of processor and disk resources in parallel computer systems, with the goal of gaining a better understanding of the impact of I/O on program behavior and resource allocation. Models were formulated to examine the I/O and computation behavior of parallel programs and workloads, and to analyze parallel resource scheduling policies under such workloads. Our analysis includes derivations of a generalization of Amdahl's law and a generalization of flow-equivalent service centers. The models and analysis were validated against measurements of parallel applications and shown to be in excellent agreement.

Our results provide important insights into the performance of parallel applications and resource scheduling strategies. In particular, our scheduling results demonstrate that, when there is contention for the disk resources, there can be significant differences in the performance trends from what has been previously observed for space-sharing policies when I/O behavior is negligible or not explicitly considered. Our results further show the potentially significant performance benefits of combining space-sharing with time-sharing to overlap the I/O of some jobs with the computation of other jobs in these environments. We are continuing to investigate the impact of workload variability and the impact of workload I/O intensity on the effectiveness of coordinated scheduling decisions and performance.

Acknowledgements. We thank the Caltech CACR for providing access to the Intel Paragon XP/S where all of our experiments were run. We thank Mark Wu and Aaron Kupperman (Caltech), and Rick Kendall and David Bernholdt (PNL) for providing the QCRD and MESSKIT codes used in our experiments.

References

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proc. AFIPS Spring Joint Comp. Conf.*, Vol. 30, pp. 483–485, 1967.

[2] S.-H. Chiang, R.K. Mansharamani, M.K. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In *Proc. ACM Sigmetrics Conf.*, pp. 33–44, 1994.

[3] N. Islam, A. Prodromidis, M.S. Squillante, L.L. Fong, A.S. Gopal. Extensible resource mangement for cluster computing. In *Proc. Int. Conf. Dist. Comp. Sys.*, 1997.

[4] E.D. Lazowska, J. Zahorjan, G.S. Graham, K.C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice Hall, 1984.

[5] C. McCann, R. Vaswani, J. Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Trans. Comp. Sys.*, 11(2):146–178, 1993.

[6] R.D. Nelson. A performance evaluation of a general parallel processing model. In *Proc. ACM Sigmetrics Conf.*, pp. 13–26, 1990.

[7] J.K. Ousterhout. Scheduling techniques for concurrent systems. In *Proc. Int. Conf. Dist. Comp. Sys.*, pp. 22–30, 1982.

[8] E.W. Parsons, K.C. Sevcik. Coordinated allocation of memory and processors in multiprocessors. In *Proc. ACM Sigmetrics Conf.*, pp. 57–67, 1996.

[9] V.G. Peris, M.S. Squillante, V.K. Naik. Analysis of the impact of memory in distributed parallel processing systems. In *Proc. ACM Sigmetrics Conf.*, pp. 5–18, 1994.

[10] D.A. Reed, R.A. Aydt, R.J. Noe, P.C. Roth, K.A. Shields, B.W. Schwartz, L.F. Tavera. Scalable performance analysis: The Pablo performance analysis environment. In *Proc. Scalable Parallel Libraries Conf.*, pp. 104–113, 1993.

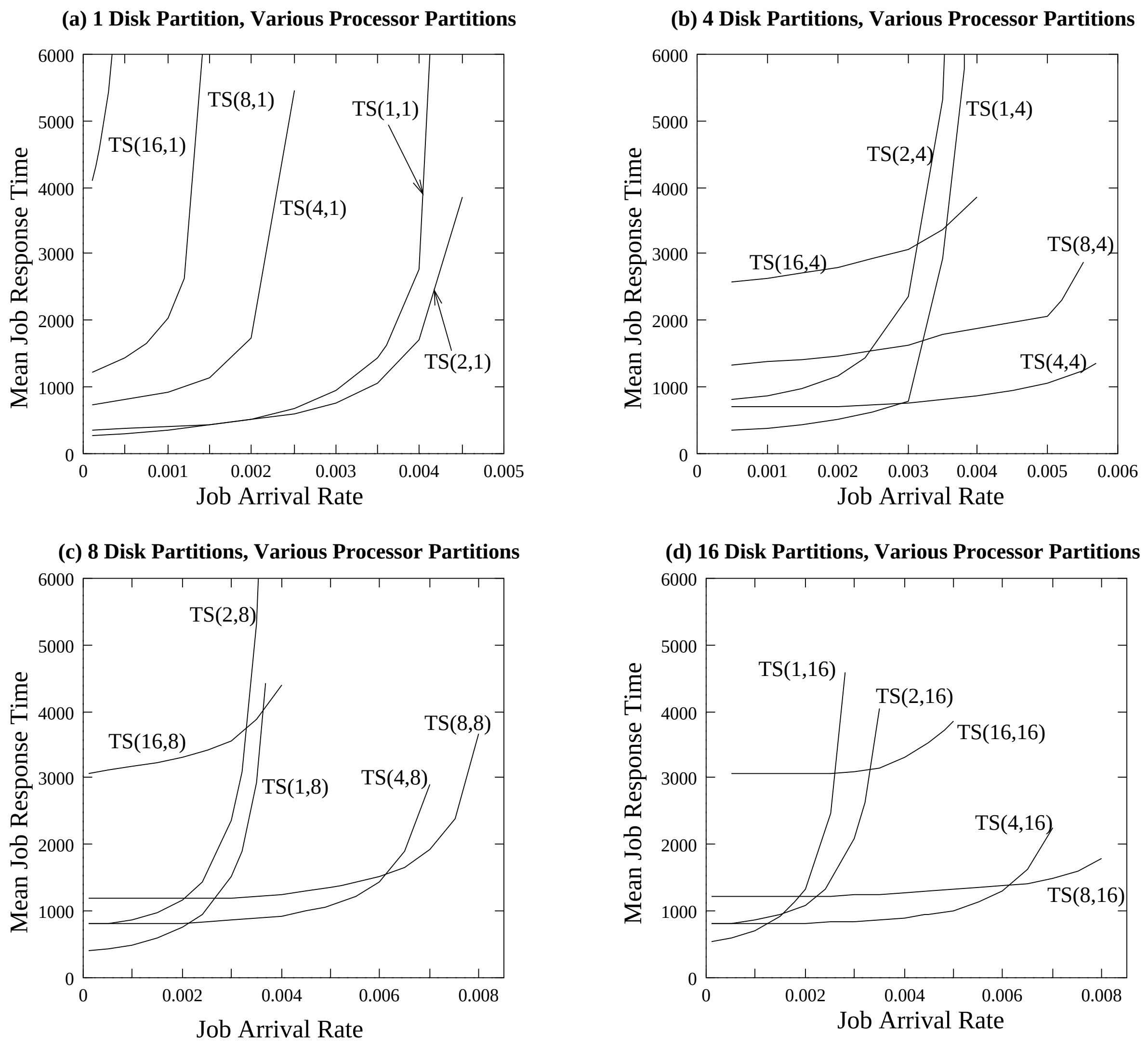


Figure 7: Mean response times under $\mathrm{TS}(K_P, K_D)$ for $K_P \in \{1, 2, 4, 8, 16\}$ and $K_D \in \{1, 4, 8, 16\}$.

[11] E. Rosti, G. Serazzi, E. Smirni, M.S. Squillante. The impact of I/O on program behavior and parallel scheduling. Tech. Rep., IBM Research Division, Sep. 1997; revised Mar. 1998.

[12] E. Rosti, E. Smirni, L.W. Dowdy, G. Serazzi, B.M. Carlson. Robust partitioning policies of multiprocessor systems. *Perf. Eval.*, 19:141–165, 1994.

[13] S. K. Setia. The interaction between memory allocation and adaptive partitioning in message-passing multicomputers. *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph (eds.), Lecture Notes in Comp. Sci. Vol. 949, pp. 146–164, 1995.

[14] E. Smirni, R.A. Aydt, A.A. Chien, D.A. Reed. I/O requirements of scientific applications: An evolutionary view. In *Proc. IEEE Int. Symp. High Perf. Dist. Comp.*, pp. 49–59, 1996.

[15] E. Smirni, D.A. Reed. Lessons from characterizing the Input/Output behavior of parallel scientific applications. To appear, *Perf. Eval.*, 1998.

[16] M.S. Squillante. On the benefits and limitations of dynamic partitioning in parallel computer systems. *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph (eds.), Lecture Notes in Comp. Sci. Vol. 949, pp. 219–238, 1995.

[17] M.S. Squillante. Matrix-analytic methods in stochastic parallel-server scheduling models. To appear, *Advances in Matrix-Analytic Methods for Stochastic Models,* S.R. Chakravarthy and A.S. Alfa (eds.), Lecture Notes in Pure and Applied Mathematics, 1998.

[18] M.S. Squillante, F. Wang, M. Papaefthymiou. Stochastic analysis of gang scheduling in parallel and distributed systems. *Perf. Eval.*, 27&28:273–296, 1996.