What is Worth Learning from Parallel Workloads? A User and Session Based Analysis

Julia Zilber
Hadassah College
37 Hanevim St.
91010 Jerusalem, Israel
juliaz@cs.huji.ac.il

Ofer Amit
Hadassah College
37 Hanevim St.
91010 Jerusalem, Israel
ofer@mail.snunit.k12.il

David Talby
Hebrew University
91904 Jerusalem, Israel
davidt@cs.huji.ac.il

ABSTRACT

Learning useful and predictable features from past workloads and exploiting them well is a major source of improvement in many operating system problems. We review known parallel workload features, and argue that the correct approach for future on-line algorithm design as well as workload modeling is user- and session-based modeling, instead of analyzing jobs directly as done today. We then provide statistically sound answers to two basic questions: Which user and session features are central enough to be potentially useful, answered using Principal Component Analysis, and which user and session classes exist and how they can be identified on-line, answered using K-means clustering. We identify variable sets that explain over 80% of the variance between sessions and between users, and also identify five stable session classes (clusters) and four stable user classes. Our analysis is based on logs from seven different parallel supercomputers, spanning over 87 months, which are analyzed together to ensure that results are location- and architecture-neutral.

Categories and Subject Descriptors

C.4 [**Operating Systems**]: Performance – *modeling and prediction*; D.4.7 [**Operating Systems**]: Organization and Design – *Distributed systems*;

General Terms

Measurement, Performance, Human Factors.

Keywords

Workload modeling, User- and session-based modeling, HPC, PCA, Clustering, User sessions' classification.

1. INTRODUCTION

Understanding the expected workload that a system will face is crucial to making the right decisions when designing and configuring it. Workload analysis for parallel computers has therefore attracted a large body of research, divided to two main flavors. The first is the construction of workload models [4,7,10,12,14], which are statistical models based on observations from real-world traces. These models can be used to create synthetic workloads, in order to compare resource management algorithms under different conditions (load or machine size, for example) or to gain general insights. The second flavor exploits features of parallel workloads directly, either by designing heuristic algorithms that exploit discovered workload features [2,17] or by designing adaptive or prediction-based algorithms that learn the workload as they go [5,6, 20,21]. Adaptive, learning and prediction-based algorithms always include a lot of prior knowledge about the workload – which parameters should be adaptive? What cues are used to change them? Which variables should be used to learn from history? – and in many cases their main innovation is discovering a particular workload phenomenon and modeling it well.

In many areas, the basic resource management policies are well-known and understood – and the major performance advances in the past few years are the result of tuning the algorithms to exploit workload features found to repeat in historic traces. Examples can be found in scheduling [10,17,20], task allocation [2,13], management of a computational grid [13], load balancing [23], soft real-time systems [6], wide-area data replication management [21] and others. All of these areas can potentially benefit from this work.

The goal of this paper is to provide new information, discovered by means of sound statistical techniques, than can substantially benefit both worlds – synthetic workload modeling, and on-line resource management algorithms. We do so by answering two very basic questions.

The first is: **which variables should be modeled?** Or in an algorithm-designer's words: which features of parallel workloads are important enough to affect performance? As the next section shows, first-generation models focused on modeling the most visible workload attributes, such as runtime and parallelism, but neglected other vital features such as the temporal structure of the workload, user behavior and so forth.

But the question is deeper: after we found these several previously unmodeled features, how do we know that this is "enough"? In other words, how do we know that a given set of variables explains most of the variance found in parallel workloads? To measure exactly that, we use Principal Components Analysis (PCA): a statistical technique used to find the important differentiating variables in a given dataset and measure the proportion of information they represent out of the total variance in the data. We also provide a concrete set of variables, which explains most of the variance between users and sessions in parallel workloads.

The second question that we address is: **what should be modeled?** Our answer is that in contrast to previous works, which modeled parallel jobs directly, we argue that users should be modeled first, then sessions, and only then jobs. The next section provides the many arguments in favor of taking this approach. This is again aimed both at synthetic models – which we believe should directly model users and sessions – as well as resource management algorithms – which can benefit more from using historical knowledge of users and sessions, than of "unattached" jobs.

In order to enable this, we analyzed a set of production workload traces, and used clustering to identify four classes of users and five classes of sessions in parallel workloads. Since we mixed several traces before clustering, this classification is both architecture- and location-neutral – and as can be observed, is mostly focused on universal human traits, such as work in sessions and the daily cycle. In addition, we provide the observed distributions of both user classes and session classes – so that together with the concrete set of variables to model given by the PCA analysis, this paper provides most of the input required to build a complete user-based workload model.

This paper is structured as follows. The next section reviews existing models, summarizes recent parallel workload features that they lack, and argues why a layered user-session-job model is the best way to embody them. Section 3 presents our dataset. Sections 4 and 5 contain the analysis of sessions, sections 5 and 6 contain the analysis of users, and section 7 summarizes a list of insights, applicable for both workload modeling and algorithm design.

2. THE CASE FOR USERS AND SESSIONS

2.1. Review of Current Workload Models

When reviewing what research has learned from parallel computer workloads so far, the obvious place to start is existing models. Several synthetic parallel workload models have been proposed to date; all directly model jobs. They differentiate one another by the method used for distribution fitting, and sometimes by the modeled parameters of the workload as well.

The first model was proposed by **Feitelson** in '96 with modifications in '97 [10]. This model is based on observations from several workload logs. Its main features are the hand-tailored distribution of job sizes (i.e. the number of processors used by each job), which emphasizes small jobs and powers of two, the correlation between job size and running time, and the repetition of job executions. In principle such repetitions should reflect feedback from the scheduler, as jobs are assumed to be re-submitted only after the previous execution terminates.

The model by **Downey** is based mainly on an analysis of the SDSC Paragon log [7]. It uses a novel log-uniform distribution to model service times (that is, the total computation time across all nodes) and average parallelism. This is supposed to be used to derive the actual runtime based on the number of processors allocated by the scheduler. In a rigid variable partitioning machine, the average parallelism can be used as the number of processors, and the service time can be divided by this number to derive the runtime.

Jann's model is based on a careful analysis of the CTC SP2 workload [12]. Both the running time and inter-arrival times are modeled using hyper Erlang distributions of common order, where the parameters for each range of number of processors are derived by matching the first three moments of the distribution. The same framework was used again to match parameters for the ASCI Blue Pacific computer.

The model by **Lublin** [14] is based on a statistical analysis of four logs. It includes a model of the number of processors used which emphasizes powers of two, a model of running times that correlates with the number of processors used by each job, and a model of inter-arrival times. While superficially similar to the Feitelson models, Lublin based the choice of distributions and their parameters on better statistical procedures in order to achieve a better representation of the original data.

The last and most recent model is by **Cirne and Berman** [4]. This is a comprehensive model for generating moldable jobs, based on the analysis of four SP/2 logs. It is composed of two parts: A model for generating a stream of rigid jobs; and a model for turning rigid jobs into moldable ones. The model also addresses the issues of requested times for a job and job cancellations. The model takes into account workday cycles, and its inter-arrivals pattern can be adjusted to match each of the logs used to build it.

A comparative study [18] has found that when comparing the models and production logs together, using only variables that the models addressed, then each model models well one or two production logs – usually the ones it was based on.

2.2. Review of Unmodeled Workload Features

There are two sources for workload features not yet found in models – statistical analysis research and algorithm research. This section summarizes what both of them have found to date.

Self-Similarity. [18] has shown that all of the major attributes of parallel workloads, including runtimes, parallelism, CPU time and number of jobs, are self-similar in nature. This is a statistical property that manifests itself in several mathematically equivalent ways: non-convergence of the load even across long time scales (weeks or months), variance that drops slowly, i.e. according to a power law rather than exponentially (so extreme cases are likely to appear), and a long-range dependence between jobs. The adaptive scheduler in [17] is motivated by stabilizing the quality of service of different months, which is caused by a mix of self-similarity and schedulers' dynamics.

Locality of Sampling. As defined in [9], this feature identifies the fact that while the overall variance in parallel workloads is very high, the diversity over short time scales, such as one day, is very small: on a typical day, at most several users are active, and they run the same jobs over and over again. Although not rigorously defined, this property is the basis of several prediction-based algorithms [17,20], which exploit the fact that the recent past of a user is a very good indicator of the (very) near future.

Daily and Weekly Cycles. Some of the models, particularly the newer ones [4,14] do reflect this feature, which is very evident in production logs [22]. A recent adaptive algorithm [17] exploits it, and shows consistent improved performance if and only if it is in sync with one of the two cycles. Older models simply cannot be used to assess such algorithms, thus losing their main use.

Flurries. A flurry [19] is a burst of extremely high activity, by a single user and over a short period of time. Flurries exhibit activity that is orders of magnitude higher then the norm during their lifetime, which usually has a significant effect of the statistics of an entire year. Flurries are essentially a kind of outlier sessions, and sessions are the only direct way to model them. Their effect on resource management algorithms is obvious.

User History. Prediction-based algorithms, aiming to predict job runtimes [5,6,13], communication patterns, load distribution across nodes [6,23], data access patterns [21] and so forth, normally rely on the history of the job's user to make predictions. A recent scheduling algorithm [20], for example, has shown substantial performance gains by combining a new backfilling scheme with user-based runtime prediction, that is based on averaging the user's last few jobs (note that locality is used as well here). As a result, existing models cannot be used to test this new algorithm at all – since none of them outputs the user of generated jobs.

Sessions. Although an intuitive aspect of human use of computers [1], sessions have not been reportedly analyzed or used so far in parallel resource management algorithms. This paper will hopefully open the door to such algorithms, by providing a clear definition, distributions and guidelines about what useful information can be learned from looking at specific sessions in addition to users. We can cautiously say that early measurements with improving the backfilling scheduler mentioned above to use sessions when predicting runtimes do show improved performance. Regardless, as the next section shows, sessions play a role in solving other problems as well.

Some algorithms learn by collecting history on specific applications [15]. However, most of the production logs don't include executable information, and its semantics is often unclear (use of command-line arguments, different compilations of the same program and so forth). Therefore, due to this lack of reliable data, we prefer to focus on users at this stage.

2.3. Users and Sessions

Our framework for parallel workload modeling and history-based algorithm design is multi-scale [3,15], and is based on users and sessions. From the modeling point of view, this means that we don't model distributions of jobs, but instead model distributions of user classes. For each user class, we model distributions of session classes; and for each session class, we model jobs. To make things simple, the session classes are identical for all user types. In the next section we uncover them independently of user types – so the only difference between user types is the frequency of each session class.

At the bottom line, such a model is used to generate a stream of synthetic jobs, like the existing models reviewed earlier. But from the algorithm-design point of view, its use is very different: during the execution of an on-line algorithm, the list of active users and (if the time scale is small) the list of active sessions are known. Therefore, only the jobs need to be simulated, and since intra-session variance is very small, the algorithm has a much more precise prediction of its near-future workload, compared to a job-based model in which distributions usually have extremely high variances. This is the first major advantage of a user-session-job model.

Second, such a model is expected to be self-similar. There are two ways to synthesize a self-similar time series: directly or by aggregation. The direct method produces signals based on a self-similar distribution, such as fractional Brownian motion or fractional ARIMA processes. These processes allow good control over the Hurst parameter, but they make it harder to control other desirable properties of the workload. The second method to produce self-similarity is by aggregating signals generated by a set of independent sources – intuitively, users – under the following conditions: users must switch between active and inactive periods – intuitively, sessions – and the duration of the "on" and "off" times must be from a heavy-tailed distribution. A distribution is called "heavy-tailed" if it has infinite variance; informally, this means that there is "significant" probability for seeing values that are extremely far from the mean. Therefore, it seems likely that basing the workload model on an aggregation of users, each exhibiting an on/off behavior (sessions), will result in a self-similar model, as long as heavy-tailed distributions are used.

Third, a user/session model provides a simple way to model time-dependent features, such as locality of sampling and cycles. Locality is very hard to model using a distribution on all jobs, since while the local variance is very small, the overall variance is very high. Users and sessions make it simple, by capturing that high variance and separating the intra-session low-variance model in a natural way. Cycles are also easy to model, due to a similar kind of separation of concerns: the inter-arrival time of jobs within a session (which doesn't have cycles) is modeled separately from the inter-arrival time between session, which obeys the daily and week cycles. We have also included locality and the known cycles as features used to define user and session classes, to ensure that our model will capture the differences between sessions that happen at different times (day versus night sessions, for example). These features were found to be of major importance in defining session classes.

Fourth, a user/session model captures flurries, as outlier sessions. Flurries showed immediately in our analysis as outliers, and a full model can be configured to either include them, as a special kind of session, or not at all. This gives researchers the choice, which is vital [19] since on one hand flurries are not characteristic of common workload, but on the other hand most long production traces contain at least one.

Fifth, since algorithms that rely specifically on users exist, then modeling users is necessary – since a main use of models is to compare competing algorithms. We believe that this need will increase, as more new algorithms will rely on user history.

And sixth, a user/session model is beneficial to understanding and exploiting workload features because it works: As the next sections will show, we succeeded in identifying the core variables that explain most of the variance between users and sessions, and then identified a small number of user and session classes that cover their entire spectrum. This proves the whole approach to be viable in practice.

3. THE DATASET

Our work is based on seven production logs from the Parallel Workloads Archive [16]. Hence, these logs have all been thoroughly checked and cleaned from many different kinds of noise, extreme outliers and errors - all very common problems with production logs. The logs contain more than 700,000 jobs that span over 87 months, and come from four different locations and five architectures.

Table 1: Parallel Computers in Our Data Set

Machine	# Nodes	#Jobs	Period
SDSC Paragon	416	32,136	Dec 94 – Dec 95
KTH IBM SP/2	100	28,490	Sep 96 - Aug 97
CTC IBM SP/2	512	77,222	Jun 96 – May 97
SDSC IBM SP/2	128	73,496	Apr 98 – Apr 00
LANL CM-5	1024	122,055	Oct 94 – Sep 96
LANL Origin 2000	2048	121,989	Nov 99 – Apr 00
SDSC Blue Horizon	1152	250,440	Apr 00 – Jan 03

To the best of our knowledge, this research is the most comprehensive to date on parallel workload modeling with respect to the number, size and quality of the logs used for the analysis. Moreover, today there are probably no other publicly available logs that can be used for such an analysis. While all seven logs come from typical high-performance computing centers, there are obvious differences caused by architectures, user population, and technical and administrative policies. In order to provide insights that are location- and architecture-neutral, which is our major goal, we analyzed **all logs together**. We extracted session and user statistics from each log, but then combined all sessions to a single list for the PCA and clustering analyses. All users were combined to a single list in the same manner. This ensures that log-specific features will disappear in the cumulative lists, and the main features left will be those that are universal to users of massive parallel computers.

4. PRINCIPAL COMPONENTS OF SESSIONS

4.1. Sessions Defined

A session is intuitively a period of continuous work of one user. This does not mean that jobs of that user are active 100% of the session's time – a user may run a job to completion, think about the result, and run another job, all within the same session. The time between the completion of the previous job and the submission of the current job is called the **think time** of the current job. Intuitively, jobs are considered to be within a single session if there is a short think time between them. There is no other formal definition of a session, and no widely accepted think time that is considered as a session boundary [1]. In order to decide what the boundary should be, we checked the cumulative distribution of think times in the logs.

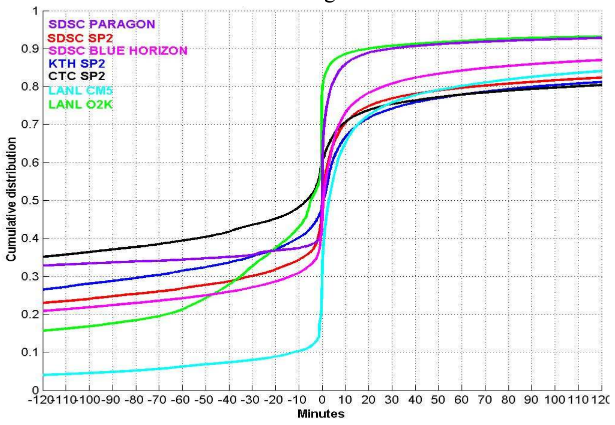

Figure 1: CDF of Think Times

The first observation from figure 1 is that the different logs are strikingly similar. This is the main reason that enabled us to analyze all logs together and draw meaningful conclusions. The second observation is that most work is indeed done in sessions: while some think times are very high (not within the same session) or way below zero (next job started before previous one was finished), most of the jobs have think times around zero – within the same session.

Following figure 1, we defined the session boundary to be twenty minutes. This is inspired by the fact that for all logs, around that time the CDF stops its steep climb and returns to a steady rise – meaning that the number of jobs started after 25, 55 or 85 minutes after their previous jobs is about the same. We took this as a cue that above this imaginary boundary the dominating distribution is that of session inter-arrivals, and not the intra-session one.

Twenty minutes is obviously not the only possible choice, but we are confident in it for two reasons. First, we've done a sensitivity analysis, by repeating the analyses using 15-minutes and 30-minutes boundary values, and received virtually the same results. Second, from a practical point of view, this choice works: Using this definition we received stable and consistent PCA and clustering results, which are useful for future modeling and algorithmic research.

4.2. Variables Set

The variables that we used in our analysis are divided to traditional ones, focused on the size of the incoming workload, and newer ones, measuring aspects of the workload's temporal structure. The traditional variables are summarized in table 2. The median and interval were preferred over the average and standard deviation, which have been shown to be highly unstable in parallel workloads due to heavy tails [8].

Table 2: Workload Variable Definitions

Symbol	Description
J	Number of Jobs
D	Duration
Rm	Median of Runtime
Ri	90% Interval of Runtime
Pm	Median of Parallelism
Pi	90% Interval of Parallelism
Im	Median of Inter-Arrival Time
Ii	Interval of Inter-Arrival Time
Tm	Median of Think Time
Ti	90% Interval of Think Time

There are three kinds of temporal structure aspects to represent: locality of sampling, daily cycle and weekly cycle. Locality of a session is represented by two simple variables: the number of unique job sizes (UP) of jobs in that session, and the number of unique runtimes (UR) of jobs in that session, where jobs are considered unique if there is a 5% difference. For most sessions these numbers are very small, which implies locality: they use only a fraction of the overall parallelism and runtime distributions.

The daily cycle is represented by two variables: a binary variable which equals 1 if the session started during the day and 0 otherwise (?D), and a continuous variable measuring the percent of the session that occurred during daytime (%D). The definition of daytime has been derived from the figure 2, which shows the distribution of job arrivals during the day (corrected for time zone shifts), for all logs.

The results are surprisingly similar for all logs –probably because the daily cycle is a universal human trait, and not a technical one. Based on figure 2, we defined daytime to be between 7:30 and 17:30.

The weekly cycle is represented by two similar variables: a binary variable that equals 1 if and only if the session started during a weekday (?W), and another one that measures the percentage of the session done during weekdays (%W). According to figure 3 (in which day number 1 is Sunday), we defined workdays to be Monday to Friday.

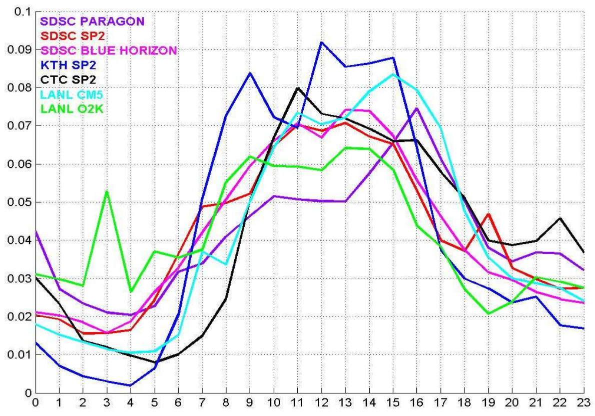

Figure 2: Job Arrivals across Hours of the Day

4.3. Principal Components Analysis

Principal component analysis (PCA) is a statistical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called **principal components**. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. The objectives of PCA are to identify patterns in data of high dimensionality, and to discover or reduce its dimensionality.

For a full presentation of PCA, see [11]; here we'll provide a short summary. Given a matrix p of observations – for example, a row for each session – we normalize it, compute the covariance matrix, and calculate the eigenvectors and eigenvalues of the covariance matrix. We then sort the eigenvectors by decreasing eigenvalues – the ones with the highest eigenvalues are the principal components. The size of each eigenvalue is proportional to the percent of variability in the original data its corresponding eigenvector captures.

The original data can be transformed to uncover principal variables by multiplying the sorted eigenvectors matrix (called a feature vector) by the transposed matrix of original data. The first columns of the resulting matrix will contain the principal component values of the data for each observation.

Our input matrix for sessions has a row for each of the **145,582 sessions** (in all logs combined), and 16 columns, one for each variable defined in section 4.2. Our focus is on finding the dimensionality of the data – hopefully discovering that a select subset of our variables is enough to explain most of the variance between sessions in parallel workloads.

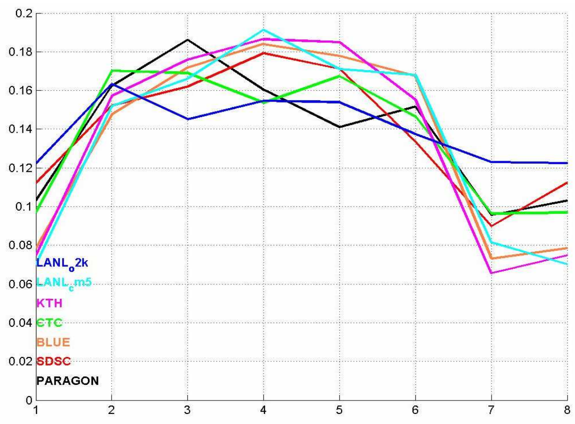

Figure 3: Job Arrivals across Days of the Week

A positive answer to this question would mean that the variables we chose indeed capture most of the variance between sessions. This is the most important point in using PCA – in contrast to some statistical techniques, the wrong variables won't produce arbitrary results that can be misinterpreted. We have experienced this over a long period of time during this research – starting only with the 12 traditional workload variables from table 2, many eigenvalues were near equal, and almost no dimensionality reduction was possible. The gradual addition of the temporal structure variables raised the percentage to their current level, in which the first 8 of 16 eigenvectors capture 85% of the variance.

4.4. PCA of Sessions

Table 3 describes the nine largest eigenvectors of the sessions' PCA analysis; the last line contains the weight of each vector – its eigenvalue divided by the sum of all eigenvalues – and the cumulative weight.

The next step is to interpret what each of the vectors means in terms of the original variables. To do so, the largest coefficients of each vector are highlighted. For example, the second vector gives most of its weight to the weekly cycle – the coefficients of the other variables in that vector are negligible in comparison. Analogically, the third vector is focused on locality, the fourth on the daily cycle, the fifth and seventh on parallelism, and the sixth on inter-arrival time.

The first – most important vector – contains several variables of similar coefficients. The reason is that these variables are correlated: The (linear) correlation coefficients between Tm and Ti is -0.75 (!), between Ti and Ri is 0.41, between Ti and Ii is 0.38, and between Ii and D is 0.42. Other pair-wise correlations of these variables are high as well, and match the findings in [18].

Table 3: Principal Eigenvectors of Sessions' PCA

	#1	#2	#3	#4	#5	#6	#7	#8	#9
J	-.09	.11	-.36	-.01	.40	.27	-.45	-.17	.17
D	-.34	-.02	.11	-.04	.03	.21	-.26	.00	-.71
Rm	-.26	-.09	.22	.06	.10	-.18	.01	-.69	-.25
Ri	-.34	.01	.00	-.05	.10	-.19	.35	-.42	.40
Tm	.39	.02	-.12	.07	-.18	.36	.16	-.37	-.06
Ti	-.44	.00	.07	-.09	.14	-.33	-.06	.33	.12
Im	-.28	-.02	.17	-.06	-.23	.56	.15	.03	.22
Ii	-.39	-.01	.16	-.08	-.16	.42	.12	.11	.12
?D	.08	.03	.05	-.70	.00	-.04	-.05	-.15	-.04
%D	.14	.05	.03	-.69	.00	.00	-.02	.01	.02
UP	-.17	.18	-.50	-.06	-.17	-.08	.30	.01	-.21
UR	-.17	.16	-.50	-.03	.27	.16	-.06	-.03	.08
?W	.04	.67	.21	.06	.05	-.01	.02	-.01	-.01
%W	.03	.67	.22	.06	.05	.00	.02	-.01	-.02
Pm	-.08	.06	.00	.05	-.56	-.13	-.66	-.19	.29
Pi	-.16	.16	-.36	-.04	-.52	-.19	.12	.01	-.17
%	19	12	12	11	8	8	6	4	4
C%	19	31	43	54	62	69	75	81	85

The intuition behind these correlations is clear: Think time is computed by summing runtimes and inter-arrival times, and high runtimes or inter-arrival times imply higher session duration. As a result, these variables' weights are correlated as well, as seen in the first vector. However, there is still a difference between the first vector, which is focused on duration, and the eighth vector, focused on runtimes (note how the duration and inter-arrival variables are negligible there).

High linear correlations explain the pairing of variables in the other vectors as well: the correlation between ?W and %W is 0.93, between ?D and %D is 0.74, between UP and UR is 0.45, between Pm and Pi is 0.24, and between Im and Ii is 0.61. PCA captures these correlations by placing correlated variables in the same vectors; this allows us to relate to the "feature" each vector represents, rather than perfecting the way each feature is measured, or making sure that it's measured once (and that variables are uncorrelated). In contrast, this will be an issue in the clustering analysis in the next section, where highly correlated variables will be filtered out from the analysis.

To verify the stability of the results in table 3, we repeated the analysis several times, slightly varying the variables set each time. The resulting vectors and the corresponding features they represent always stay the same; however, the order of the vector may change. For example, if only 1 out of 16 variables measures locality (for example, if UR is removed), then the locality vector would move from 3rd vector to 5th place, because the locality feature is now less evident in the dataset. However, the main features and the proportion of explained variance is about the same in all analyses.

To conclude, we found that the following features explain most of the variance between sessions:

- **Interval of Inter-arrival / think times**
- **Weekly cycle**
- **Locality**
- **Daily cycle**
- **Parallelism**
- **Inter-arrival time**

Arguably, these variables alone are not enough, since although they dominate certain eigenvectors, there are also non-zero coefficients in each eigenvector, which are required to build it, and ignoring them spoils the results. However, this is more than compensated in the seven smallest eigenvectors (not shown in table 3), all of each are also dominated by the above variables.

Note the high dominance of the temporal structure variables, which synthetic models to date have largely ignored. In contrast, the runtime seems to play a surprisingly minor role.

5. CLUSTERS OF SESSIONS

5.1. Methodology

Identifying and characterizing a small number of consistent session clusters is of high practical importance to both algorithm design and workload modeling. We will use the classic K-means clustering algorithm [11], which in a nutshell works by iterating until convergence a two step process: compute estimated centers of clusters, and tag each observation to belong to the cluster to which it is closest.

The algorithm requires the number of clusters as input, and finding the "right" number of clusters is highly problem-specific and sometimes subjective [11]. We have experienced with a large number of clustering results (a practice required anyway to verify the stability of our results), and concluded that using five clusters gives the most stable and useable results.

Another methodological issue is the variables set by which the clustering is performed. In contrast to the PCA analysis in which we used all candidate variables, here it is desirable to remove highly correlated variables, so that each feature is represented once and the algorithm is not distorted to cluster by any one particular feature. The variables used for the clustering shown here are Duration, Think-time interval, Day time part, Work week part, Unique processors count, Parallelism median and Runtime median.

5.2. The Five Session Classes

Table 4 shows the mean, median and interval of each variable in each cluster. Figure 5 shows some of the full distributions of sessions, ordered and colored by cluster using the same colors of table 4 and figure 4. Analysis of the data confirms that the clusters correspond to intuitive session types: interactive versus batch work, day versus night, and weekday versus weekend. This enables giving each cluster a significant name.

Interactive, workday, daytime sessions. This is the most common session class (43%). It occurs always in weekdays and starts during daytime in 95% of its sessions. Each session has very few jobs (median is 2.0 and mean 3.4), resulting in high locality. Runtimes and parallelism are low, typical for interactive work.

Interactive, workday, nighttime sessions. These 29% of the sessions are active mostly during the night (82%) although only 63% start at night. The number of jobs, inter-arrival times and think times are short and typical of interactive work; locality is high as well. On the other hand, runtimes are much higher (median of 33 minutes in contrast to 6 minutes in the daytime cluster). The most plausible explanation is that these sessions often represent someone who works interactively during the day, and towards the evening starts one or more long job that run during the night.

	Int. Workday Daytime			Int. Workday Night			Interactive Weekend			Batch Highly Parallel			Batch High Duration		
	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int
J	3.4	2.0	10.0	5.0	2.0	15.0	4.2	2.0	12.0	2.5	1.0	6.0	22.4	6.0	48.0
D	5634	2626	28371	45445	16452	179584	19560	4186	93611	55788	20713	237677	363495	186533	1246071
Rm	2418	383	12006	11322	2008	53114	5076	485	29682	8068	2774	34182	24304	10828	92299
Ri	2265	9	12821	5583	18	37010	3723	14	24118	3786	0	19284	32825	18104	119838
Tm	-1123	0	6948	-6101	0	40581	-3311	0	21876	-5863	0	34675	-103705	-40434	396074
Ti	4035	112	23819	13423	155	75626	8620	161	53907	13782	0	83244	259850	200832	750768
Im	428	32	1966	1160	17	5751	701	28	2921	2086	0	10984	19722	420	102109
Ii	1435	136	7529	6264	87	41378	3126	128	14006	7822	0	44333	90006	32286	344532
?D	0.95	1.00	0.00	0.37	0.00	1.00	0.72	1.00	1.00	0.67	1.00	1.00	0.66	1.00	1.00
%D	0.97	1.00	0.27	0.18	0.00	0.52	0.65	1.00	1.00	0.55	0.46	1.00	0.47	0.42	1.00
UP	1.28	1.00	2.00	1.27	1.00	2.00	1.30	1.00	2.00	1.25	1.00	1.00	3.33	2.00	7.00
UR	2.49	2.00	6.00	2.75	2.00	7.00	2.63	2.00	7.00	1.93	1.00	4.00	6.55	4.00	17.00
?W	1.00	1.00	0.00	0.95	1.00	0.42	0.02	0.00	0.16	0.78	1.00	1.00	0.73	0.76	1.00
%W	1.00	1.00	0.00	0.95	1.00	1.00	0.04	0.00	0.00	0.79	1.00	1.00	0.77	1.00	1.00
Pm	4.99	3.56	15.94	5.54	2.56	21.27	4.32	1.78	15.94	61.76	64.00	60.62	9.59	6.00	31.38
Pi	1.77	0.00	10.24	1.95	0.00	12.00	1.80	0.00	10.00	9.52	0.00	56.89	13.54	5.29	62.00

Table 4: Characteristics of the Five Session Classes

Interactive, weekend sessions. The weekly cycle is the next important differentiator between sessions, according to these results. 98% of sessions in this cluster start during weekends. The statistics are typical of interactive work, and are mostly in between the values of the previous two weekday clusters.

Batch, highly parallel sessions. The last two clusters obviously represent batch work, and are divided between parallelism and runtime. Both clusters are active near-equally at day vs. night and weekday vs. weekend – note that as we defined it daytime is 42% of each 24-hour day, and workdays are 71% of a week. This cluster has sessions with usually one job (median 1.0, mean 2.5), very high parallelism (median 64.0, meaning half the machine since it's normalized across logs), and much higher runtimes than the interactive sessions.

Batch, high duration sessions. These batch sessions have higher parallelism than the interactive sessions (5.29 median), and most evidently – a runtime median of 51.8 hours and mean of 100.9 hours, hinting that the runtime distribution has a long tail. This is in sharp contrast to a median runtime of 5.75 hours in the other batch sessions cluster, and much less in the interactive ones. This session class has the most jobs (median 6.0, mean 22.4), but this may be caused in part by the way we defined sessions using think times.

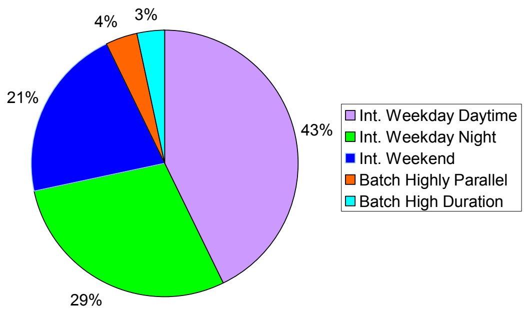

Figure 4: Distribution of Session Classes

6. PRINCIPAL COMPONENTS OF USERS

6.1. Variables Set

Our data has **2,048 users** in all the logs combined. The traditional variables were kept, computed on all the jobs on each user: median and 90% interval of runtime, parallelism, inter-arrival time and think time. The total number of jobs J and the total number of sessions S are used to quantify the user's activity. To measure how continuous and intensive that activity was, a jobs per week (JW) variable was added; we also computed a sessions per week variable, but its linear correlation to JW was full (1.0), so it was dropped. Duration D in days was not used in the PCA analysis, but was measured and will be given in the table of clusters.

Some of the temporal structure variables make no sense to measure at the user level – the UP, UR, ?D and ?W variables lose their meaning when jobs from different sessions are combined. The %D and %W variables over each user's job were measured, as they describe the user's habits outside a single session scope.

The correct way to analyze users' temporal behavior, including aspects such as locality, is to take advantage of our analysis of sessions. For each user, we add five more variables, each counting the proportion of each session type in that user's sessions. This encapsulates many aspects of the user's work patterns, and as the analysis will show, provides good results. The variable names correspond to the initials of the session classes' names from section 5.2, and in the same order are %IWD, %IWN, %IW, %BHP and %BHD. Note that the proportion of session of each class is used and not the number of sessions, to decouple the amount of work (measured using the J and S variables) from the measurement of work patterns.

The first nine eigenvectors of the PCA analysis of the above 18 variables is given in table 5. The two think time variables (Tm, Ti) are absent, since they have a correlation of and 1.0 and 0.99 with the Im and Ii variables and therefore have exact similar values.

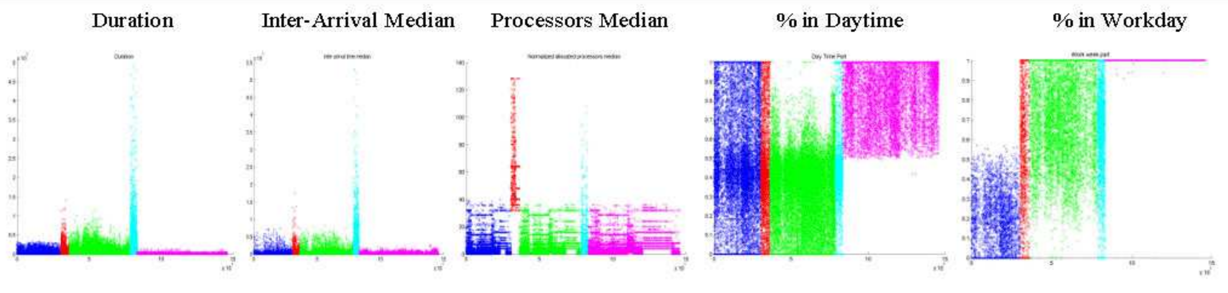

Figure 5: Distribution of Selected Variables by Session Cluster

6.2. PCA of Users

The first 9 out of 18 eigenvectors in the users' PCA capture 83% of the variable between users. However, the next five eigenvectors are a repetition of eigenvectors 5,6,2,3 and 4 respectively – meaning they have the same dominant variables – so with the same set of variables used for the first eight eigenvectors, 96.7% of the variable is captured.

As with sessions, each eigenvector represents a feature, and variables are grouped by correlation. These features explain most of the variance between users:

- **Daily Cycle**
- **Parallelism**
- **Runtime**
- **Number of jobs / sessions**
- **Weekly cycle**
- **Inter-arrival time**
- **Jobs per week**

	#1	#2	#3	#4	#5	#6	#7	#8	#9
J	-.01	.13	-.01	-.64	.06	-.26	.15	-.06	.04
S	.04	.19	.00	-.62	.05	-.23	.03	.20	.01
Rm	-.24	.07	-.44	.12	.30	-.07	.06	.36	-.16
Ri	-.20	.16	-.50	.07	.22	.00	.07	.22	-.01
Pm	-.08	.51	.18	.16	-.04	.00	.02	.13	-.07
Pi	.01	.50	.13	.03	-.07	.05	-.05	-.17	.05
Im	.01	.01	-.08	.19	-.25	-.63	.21	-.04	-.12
Ii	.09	-.02	-.13	.25	-.26	-.54	.06	-.09	.25
%D	.49	.11	-.28	.08	.10	.06	.12	.18	-.03
%W	.07	-.02	.20	.08	.47	-.32	-.32	-.22	-.66
%IWD	.57	-.05	.08	.05	.25	-.02	.13	.10	.13
%IWN	-.53	-.19	.18	.01	.10	-.13	-.10	.13	.19
%WD	.01	-.05	-.35	-.17	-.59	.17	-.16	-.08	-.51
%BHP	-.04	.54	.21	.15	-.14	.02	-.03	.20	-.07
%BHD	-.14	.23	-.30	.01	.23	.09	.26	-.75	.11
JW	-.12	-.13	.26	.03	-.02	.12	.83	.08	-.36
%	16	15	11	10	8	7	6	5	5
C%	16	30	41	51	60	67	73	78	83

Table 5: Principal Eigenvectors of Users' PCA

7. CLUSTERS OF USERS

Users were clustered with the same methodology used for session clustering. The variables selected to perform the clustering are Day time part, Parallelism median, Runtime median, Number of jobs, Work week part, Inter-arrival time median, Jobs per week, and the proportion of batch high-duration sessions. These variables include the dominant coefficients in all eigenvectors of the users' PCA analysis. The best results were obtained with four clusters, and as with sessions, it is possible to assign meaningful names to each user class, corresponding to intuitive user types.

Long-term, Light users. 55% of users belong to this class, whose members have 26 sessions over a period of 125 days. These are the medians – the means are much higher, indicating a long tail of the respective distribution. According to the proportions of session classes for these users, their focus on interactive work is higher than the overall average – this class seems to represent people whose day job involves use of the parallel computer. The runtimes, parallelism, inter-arrival times and number of jobs per week are all in accord with a mainly interactive style of work.

Long-Term, Heavy users. These 20% of the users are the source of most of the load on the computer. 6% of their sessions are BHP and 13% of their sessions are BHR, in contrast to 1-2% in all other user classes. These users produce most of the sessions of these two batch classes. These users are also the heaviest users of the machine in terms of number of jobs, sessions and duration, by a significant margin. They work both day and night, workday and weekends in equal proportions. Their runtime, parallelism and inter-arrival statistics are high, a mix of their interactive and batch sessions.

Short-term, Weekend users. The last two user classes represent users who worked on the computer for a short period – a median of 4 sessions, with median durations of 12 and 7 weeks, and a small total number of jobs (8 and 10, although the means are much higher). These seem to be users who received access to the computer for one computationally demanding project. The two user classes differ by their temporal work patterns, as measured by %D, %W and the session classes' proportions. The third "weekend" cluster, consisting 10% of the user population, has 78% of jobs starting on weekends, and the proportions between the three interactive session classes are much closer than in the overall distribution of session classes.

Short-term, Workday users. 15% of the users belong to this fourth cluster, composed of short-term users with two strong habits: they prefer workdays over weekends (90% of jobs ran in workdays), and they prefer the night over day time (82% of jobs ran during the night). This is reflected by the fact that 65% of their sessions are interactive workday night sessions.

Both short-term user classes consist mostly of interactive work, reflected by low runtime and parallelism statistics, and very intensive work, reflected by a very high number of jobs per week.

	Long-Term Light User			Long-Term Heavy User			Short-Term Weekend User			Short-Term Workday User		
	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int
J	259	68	1252	526	171	1529	48	8	242	170	10	683
S	70	26	281	82	50	223	12	4	47	26	4	117
Rm	2,569	241	13,092	12,292	4,668	48,951	1,653	91	9,679	1,984	107	10,822
Ri	11,853	5,269	43,228	54,460	56,064	117,519	7,609	821	43,205	3,750	256	23,333
Pm	3.22	1.78	14.94	8.45	3.56	31.75	2.98	1.00	10.18	3.20	1.25	10.18
Pi	11.06	3.75	60.00	18.31	8.98	64.00	5.40	0.19	30.00	5.30	0.00	28.00
Im	33,349	390	71,072	-7,063	6	125,444	50,996	147	29,381	21,784	0	21,522
Ii	1,135,229	261,020	5,025,95	776,389	396,090	2,684,896	1,058,137	119,341	4,899,134	471,280	34,469	2,499,302
Tm	39,299	1,419	88,965	31,567	2,659	133,638	56,819	548	69,745	24,742	204	38,180
Ti	1,134,561	256,335	5,023,631	693,984	283,564	2,340,145	1,060,989	106,129	4,899,777	471,995	28,530	2,497,021
%D	0.82	0.86	0.47	0.66	0.67	0.61	0.60	0.67	1.00	0.18	0.18	0.48
%W	0.80	0.78	0.46	0.72	0.72	0.42	0.22	0.25	0.50	0.90	1.00	0.33
%IWD	55%	51%	83%	24%	23%	48%	36%	30%	100%	17%	10%	50%
%IWN	22%	20%	53%	38%	38%	69%	36%	25%	100%	65%	60%	100%
%WD	20%	20%	44%	19%	19%	38%	25%	10%	100%	17%	3%	67%
%BHP	2%	0%	11%	6%	0%	47%	2%	0%	3%	1%	0%	2%
%BHD	1%	0%	7%	13%	8%	43%	1%	0%	7%	1%	0%	4%
JW	126.30	6.03	502.65	25.13	4.86	47.57	1139.86	10.51	6951.26	1247.26	20.48	7114.54
D(days)	239	125	789	323	260	848	97	12	583	110	7	636

Table 6: Characteristics of the Four User Classes

Figure 6 contains several graphs that visualize the difference between the user clusters. Figure 7 below summarizes the distribution of user classes.

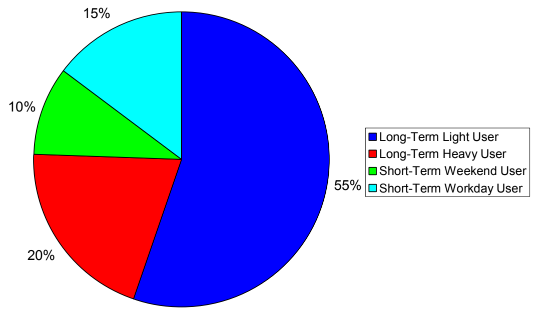

Figure 7: Distribution of User Classes

Since we have conducted all our analyses on the combined workload of seven logs, it is necessary to remap the clusters – of both users and sessions – to the original logs, to verify that we haven't clustered according to the logs. The full data is not given here due to lack of space, but the bottom-line results are that the session and user classes that we identified exist clearly in all logs, and are indeed unrelated to particular locations or architectures.

7. Using These Results

The above sections complete our search to find what is worth learning from historic parallel workloads, and to present that information in a simple and applicable way. There are two direct ways to use the results.

The first is workload modeling. The PCA analysis answers a basic methodological question – what needs to be modeled? The temporal structure variables in particular have been shown to be of great importance, while some other variables have not. The PCA analysis was also a precondition to a meaningful clustering, since it ensures us that we are clustering according to a suitable set of variables – which wasn't known before.

We have also argued for the construction of a user-based model, because it's required by recent algorithms, which make explicit use of the user field of each job, and because it's a simple way to model several newly discovered unmodeled features – some of which we found to be of vital importance.

In addition, we provided a very specific blueprint for constructing such a model: the set of variables, the user classes and their distributions, and the session classes and their distribution. To complete the model, full distribution fitting of each modeled variable in each user and session class is required, as well as a distribution for the inter-arrival times of new users

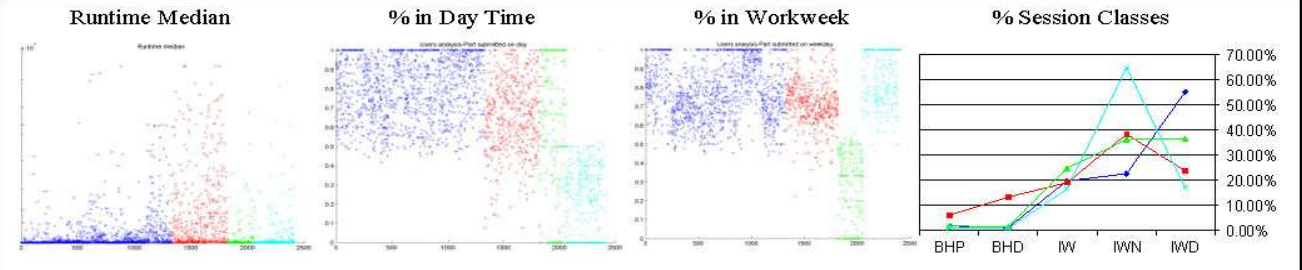

Figure 6: Distribution of Selected Variables by User Cluster

and new sessions. Building such a model and validating that it represents the unmodeled features we listed well is a desirable future research direction from this work.

The second use of this work is algorithm design. Consider for example an algorithm that relies on good runtime prediction, such as a backfilling scheduler [20], a grid management system [13] or a soft real-time task mapping service [6]. While relying on the user's history to make predictions is common practice, relying on sessions is not, and is enabled now.

When a job arrives, it needs to be attached to a session, made simple using the same-user, 20-minute think time rule. If the job is starting a new session, then its class must be determined, which is also easy: if it exceeds a certain requested runtime or parallelism – half the machine's maximum, for example – the session is tagged as BHD or BHP respectively. Otherwise, the session type is determined solely by the day of the week and the time of the day.

Once the session class is determined, we know its distribution of runtimes. This distribution is far narrower – and thus contains more informative – than what can be known a-priori by other means. Three layers of narrowing are at work here: the user level, the session level, and the session class level.

If a given job is not the first within its session, then locality enables us to predict its runtime with great accuracy, based on the past jobs of that session. For example, In the IWD session class – to which 43% of sessions belong – the mean unique number of runtimes is just 1.28. Sessions, as we defined them, are a natural way to capture and exploit locality, especially since determining a session's class is so easy. Using only the user, or the last hour or day, is open to much more noise and thus to inferior results.

This is not to say that a prediction algorithm is not necessary. But given the a-priori knowledge embodied in this work, there is much less left to learn. We believe this should lead to reassessing existing predictors, and in particular to the preference of simple "knowledge-packed" algorithms overly highly sophisticated AI techniques [13,23] that assume little in advance. This is a primary future research direction of this work.

Similar to runtime prediction is the problem of load prediction, found in load balancing [23], grid or multi-cluster scheduling [13] and soft real-time or general QoS enabled systems [6]. Here we must predict both runtime and parallelism, and sometimes the number of jobs as well. Again, our tables provide very close estimates, coupled with a clear-cut way to define a session and classify it.

Many other problems require workload prediction. Regardless of the problem and of whether the algorithm that tries to solve it is predictive, adaptive, dynamic, learning or plain heuristic – this work provides a lot of sound prior knowledge on parallel workloads it can easily use. Due to the large size of our data, the architecture-neutrality of the analysis, and the stability of the results, we believe that they can be highly useful for a large variety of applications.

8. REFERENCES

[1] M. Arlitt, *Characterizing Web User Sessions*. In *Perf. Eval. Rev.* 28(2), pp. 50-56, Sep 2000.

[2] B. Schroeder and M. Harchol-Balter. *Evaluation of Task Assignment Policies for Supercomputing Servers: The Case for Load Unbalancing and Fairness, IEEE Symp. on High Perf. Dist. Comp.*, Aug 2000.

[3] M. Calzarossa and G. Serazzi, *Construction and Use of Multiclass Workload Models*. In *Perf. Eval.* 19(4), pp. 341-352, 1994.

[4] W. Cirne and F. Berman, *A Model for Moldable Supercomputer Workloads*. Intl. Parallel & Distr. Processing Symp., Apr 2001.

[5] P. Dinda, *Online Prediction of the Running Time of Tasks*, Cluster Comp., Volume 5, Number 3, 2002.

[6] P. Dinda, B. Lowekamp, L. Kallivokas, D. O'Hallaron, *The Case for Prediction-based Best-effort Real-time Systems*, Proc. of the Intl. Workshop on Parallel and Distributed Real-Time Systems Apr 1999, pp. 309-318.

[7] A. B. Downey, *A Parallel Workload Model and Its Implications for Processor Allocation. Intl. Symp. High Perf. Dist. Comp.*, Aug 1997.

[8] A. B. Downey and D. G. Feitelson, *The Elusive Goal of Workload Characterization*. In *Perf. Eval. Rev.* 26(4), pp. 14-29, Mar 1999.

[9] D. G. Feitelson, *Workload Modeling for Performance Evaluation*. In Perf. Eval. of Complex Systems: Techniques and Tools, M.C. Calsarossa and S. Tucci (Eds.), pp. 114-141, Springet-Verlag, Sep 2002. LNCS vol. 2459.

[10] D. G. Feitelson and M. A. Jette, *Improved Utilization and Responsiveness with Gang Scheduling*, In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1997, LNCS vol. 1291, pp. 238-261.

[11] Paolo Giudici, *Applied Data Mining: Statistical Methods for Business and Industry*. Wiley & Sons, 2003.

[12] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira and J. Riodan, Modeling of Workload in MPPs. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1997, LNCS vol. 1291, pp. 95-116.

[13] S. A. Jarvis, D. P. Spooner, H. N. Lim Choi Keung, J. Cao, S. Saini, and G. R. Nudd. *Performance Prediction and its Use in Parallel and Distributed Computing Systems*. In *Future Generation Computer Systems special issue on System Perf. Analysis and Eval.*, 2004

[14] U. Lublin and D. G. Feitelson, *The workload on parallel supercomputers: modeling the characteristics of rigid jobs. J. Parallel & Dist .Comp.* 63(11), pp. 1105-1122, 2003.

[15] D. A. Menasc'e, V. A. F. Almeida, R. Riedi, F. Ribeiro, R. Fonseca and W. Meira Jr., *A hierarchical and multiscale approach to analyze E-business workloads*. In Perf. Eval. 54(1), pp. 33-57, Sep 2003.

[16] The Parallel Workloads Archive, *http://www.cs.huji.ac.il/labs/parallel/workload*

[17] D. Talby and D. G. Feitelson, *Improving and Stabilizing Parallel Computer Performance Using Adaptive Backfilling*. In *Intl. Parallel & Distributed Processing Symp.*, Apr 2005.

[18] D. Talby, D. G. Feitelson, and A. Raveh, *Comparing logs and models of parallel workloads using the Co-Plot method*. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (Eds.), pp. 43-66, Springer-Verlag, 1999. LNCS vol. 1659.

[19] D. Tsafrir and D. G. Feitelson, *Workload flurries*. TR 2003-85, School of Computer Science and Engineering, Hebrew University of Jerusalem, Nov 2003.

[20] D. Tsafrir, Y. Etsion and D. G. Feitelson, *Backfilling Using Runtime Predictions Rather than User Estimates*. TR 2005-5, School of Computer Science and Engineering, Hebrew University of Jerusalem, Nov 2003.

[21] S. Vazhkudai, J. Schopf, I. Foster,*Predicting the Performance of Wide-Area Data Transfers*. In *Int'l Parallel and Dist. Processing Symp., Apr 2002.*

[22] A. B. Yoo and M. A. Jette, *The Characteristics of Workload on ASCI Blue-Pacific at LLNL*. In Intl. Symp. on Cluster Comp. and the Grid, May 2001.

[23] Kun-Ming Yu, Wu, S.J.-W. and Tzung-Pei Hong, *A Load Balancing Algorithm using Prediction*. In Proc. 2nd Aizu I