Workload Characterization Issues and Methodologies

Maria Calzarossa, Luisa Massari, and Daniele Tessera

Dipartimento di Informatica e Sistemistica, Università di Pavia,
via Ferrata, 1, I-27100 Pavia, Italy
{mcc,massari,tessera}@alice.unipv.it

1 Introduction

The performance of any type of system cannot be determined without knowing the workload, that is, the requests being processed. Workload characterization consists of a description of the workload by means of quantitative parameters and functions; the objective is to derive a model able to show, capture, and reproduce the behavior of the workload and its most important features.

A survey of workload characterization (i. e. the parameters and the techniques used for characterizing batch, interactive, database, network–based and parallel workloads) were presented in [11].

Workload characterization dates back to early 70's. Since then this discipline has evolved following the evolution of computer architectures. In the early days, computers were mainframes and their workloads were basically composed of batch jobs and transactions. The advent of time sharing systems and computer networks has changed the approach of the users toward the systems. This advent, which has been coupled with an increased processing power of the systems and with the introduction of graphical user interfaces, has opened the systems to new processing requirements. All this has also led to the development of distributed systems and of client/server applications.

The new services provided on top of Internet, such as the World Wide Web, have introduced the concept of multimedia workloads. These workloads consist of a mix of different types of application (e.g., file transfers, real time audio applications) characterized by different performance requirements on the resources of servers and clients as well as of the networks.

Vector processors, multiprocessors, and parallel systems deserve their own attention. Their workload consists of compute–intensive and I/O–intensive scientific applications which take advantage in a variety of ways of the various processing units of the system in order to reduce their overall execution time.

In these years, workload characterization has been addressing all the new application domains. The techniques applied for this purpose have evolved accordingly to cope with the nature of the workloads which have become more complex. There are many performance studies that have to be addressed by means of workload characterization. Examples are competitive procurement, system sizing, capacity planning, performance comparisons for marketing purposes.

G. Haring et al. (Eds.): Performance Evaluation, LNCS 1769, pp. 459–482, 2000.

Another key example is represented by benchmarking. The definition of benchmark suites to be used to test both the systems currently available and new emerging systems has to rely on an accurate characterization of the workload of these systems. Moreover, the design and evaluation of resource management policies, such as caching policies for World Wide Web servers, require the knowledge of the characteristics and behavior of the requests to be processed by the servers. The design of efficient parallelizing compilers has to rely on models able to predict the behavior and the performance of parallel applications.

The paper is organized as follows. Section 2 describes the approaches commonly adopted for workload characterization. We focus on the issues to be addressed in this framework and on the various techniques used to build models which resemble either the static and dynamic properties of the workload. Sections 3 and 4 present case studies where the workloads processed in client/server environments and by parallel systems are analyzed. Future directions in the field of workload characterization are outlined in Section 5.

2 Approach

The term "workload characteristics" refers to the demands placed by the requests on the various system resources. Each request, that is, each workload component, is described by a set of parameters which explain these demands. There are static parameters related to hardware and software resource consumptions and dynamic parameters related to the behavior of the requests. The magnitudes of these parameters depend on the nature of the single request and can be conveniently used to quantify the workload.

The approach commonly adopted for workload characterization is experimental, that is, based on the analysis of measurements collected on the system while the workload is being processed. Due to the complexity of the systems and of their workloads, such an experimental approach becomes quite challenging. Appropriate instrumentation has to be developed in order to ensure the quality of the measurements which have to adjust to the characteristics of the systems and of their workloads. To capture detailed information about the behavior of the workload, it might be necessary to insert into the system probes, such as event counters. Another issue to be addressed deals with the large amount of collected measurements. A good tradeoff between what has to be measured and the amount of collected data has to be achieved. Moreover, the degree of intrusiveness and the overhead introduced by the instrumentation system have to be as low as possible in order not to perturb the behavior of the system and of its workload. Once the measurements are available, appropriate techniques have to be applied for their analysis in order to ensure the accuracy and the representativeness of the workload model which will be derived.

Several issues have to be addressed when building a workload model. Representativeness is a measure of fit between the model and the actual workload. The characteristics of a representative workload model must resemble those of the actual workload with reasonable fidelity while satisfying several practical

constraints. Modeling tends to smooth out details of the workload which might be desirable to investigate. Hence, the concepts of model abstraction and how much loss of information is acceptable have to be addressed. Another crucial point is to assess the most "important" features to be included in a model.

There are many advantages in building and using workload models. A model is typically artificial and generative, that is, it parametrically generates the workload. Hence, performance studies can be carried out without requiring actual measurements which might be either not easily manageable or not available. Moreover, portability and reproducibility requirements are fully met by workload models.

There is a large variety of techniques used for workload characterization. Exploratory data analysis techniques are fundamental to uncover the essential characteristics of the measured quantities and their relationships. Numerical analysis techniques and stochastic processes are useful for reproducing the dynamic behavior of the workload.

Descriptive statistics of workload parameters (e.g., basic statistics, correlations, distributions) provide preliminary insights into the behavior of the workload. Emphasis has also to be put on visual examination of the data. Even though the number of parameters describing the workload, that is, the data set dimensionality, and the number of data points, that is, the number of workload components, may be quite large, data visualization suggests patterns which might be not obvious from statistical summaries only. This is often the case of the workloads processed by parallel systems.

Clustering is a numerical pattern recognition technique commonly adopted in the framework of workload characterization. The goal is to find structures in large volumes of data. The workload components are represented as points in a multi dimensional space, whose dimension is equal to the number of parameters used to describe the components. Clustering partitions the data space into groups characterized by their centroids, that is, their geometric centers. The choice of the quantitative parameters to be used for clustering has to be such that a common description of the composition of each group can be easily obtained. Moreover, the choice of these parameters has to be driven by the intended use of the clustering outcomes. This is the case, for example, of the specification of the input of system models. The groups provided by clustering specify the classes for these models. The parameters corresponding to the centroid of each group are used as input parameters of system models.

It is also important to provide a description of the composition of each group from a functional view point. In such a case, qualitative parameters, such as request type, which are not advisable to use for clustering purposes, can be profitably used to describe the workload components belonging to each group. This information will drive, for example, the choice of representative workload components to be used in benchmarking experiments.

Descriptive statistics and clustering have always been widely applied to workload characterization. Early papers (see e.g., [18], [46], [2], [41]) focus on these techniques in order to find a set of representative programs to be used for bench-

marking and for system modeling. The target of these studies is batch and interactive systems, whose workload components are described by parameters, such as CPU time, number of I/O accesses, number of lines printed, amount of central memory used. Qualitative parameters, such as programming language and types of activity performed (e.g., editing, compilation, execution), are also used to combine a functional description of the workload models with their quantitative counterpart. General descriptive statistics are also used to characterize the workload of personal computers and to derive the behavior of the users (see e.g., [50]).

In most of these studies, only the static characteristics of the workload are taken into account. When the dynamic behavior of the workload has to be analyzed, descriptive approaches based on the application of graphical and mathematical methods are used. The objective is to derive workload models to be incorporated into system models, such as simulation models, or into the description of operating system policies.

There is a variety of phenomena which have to be described from a dynamic view point. In [30], statistical analysis of series of events is applied to the sequences of transactions initiated in a database system. The arrival rate of the transactions is represented by means of a non homogeneous Poisson process obtained as a superposition of a large number of non stationary point processes. Numerical fitting techniques are applied in [10] to model the fluctuations characterizing the patterns of the jobs arriving at a system. The combined application of clustering and fitting techniques leads to the identification of representative arrival patterns. In [1], sequences of job steps in a job are modeled by means of a Markov chain whose states represent the calls to the various programs (e.g., compilers, utilities). Stochastic models, based on Markov chains, are used in [23] to represent the workload of interactive systems at the task level.

Graph–based models are a common representation of the sequences of commands issued by interactive users. User behavior graphs, introduced in [19], are the most popular example of these types of model. The nodes of a user behavior graph correspond to the various types of command. The arcs, with their associated probabilities, represent the sequences of commands as issued by the users. Fig. 1 shows an example of a user behavior graph consisting of eight nodes.

The introduction of graphical user interfaces based on multiple windows has led to the development of models which have to take into account the characteristics of these operating environments (see e.g., [38]). A sort of parallelism is introduced in the user behavior, in that a user can issue many commands from different windows, without having to wait for the completion of the current one. All these commands are then simultaneously "active" in the system. Hence, workload models have to generate these sequences of commands and reproduce the parallelism characterizing their executions.

Moreover, with the advent of client/server environments, the hierarchical nature of the workload, earlier recognized for interactive workloads ([23]), has become more evident and more relevant (see e.g., [8]). Various systems are involved in processing a single request. For example, a command issued by a user

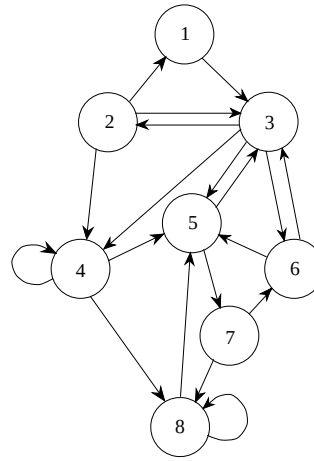


Fig. 1. User behavior graph consisting of eight nodes.

from a workstation could be processed by the local workstation only or might require, as in the case of distributed file servers, processing from remote systems, after having accessed the network. Hence, a single request is broken down into a number of sub–requests which invoke services by different systems. This means that each system "sees" different types of request, that is, a different workload. A number of layers, corresponding to the various systems which contribute to process a request, can then be identified for the characterization of the workload. Different parameters are used to describe the workload of each of these layers.

In this framework, the concept of networkload has been introduced ([39]). The networkload is a collection of user generated inputs or client generated events. Fig. 2 shows the layers (i.e., session, command, request) of the hierarchy as identified in the networkload approach. A Probabilistic Context Free Grammar is used to describe and generate these sequences. A generative model, characterized by variable levels of hierarchy, is then obtained.

Recently, notions, such as self–similarity, have been shown to apply in the framework of workload characterization ([29]). The various types of network traffic are characterized by a bursty nature and exhibit long–range dependence, which can be represented by heavy–tailed distributions. The same applies to the requests, e.g., read, write, open, close, processed by a file system ([21]). Self–similarity is able to capture and explain these phenomena.

More details about this approach and other techniques used to characterize the workload will be described on the case studies presented in the following sections.

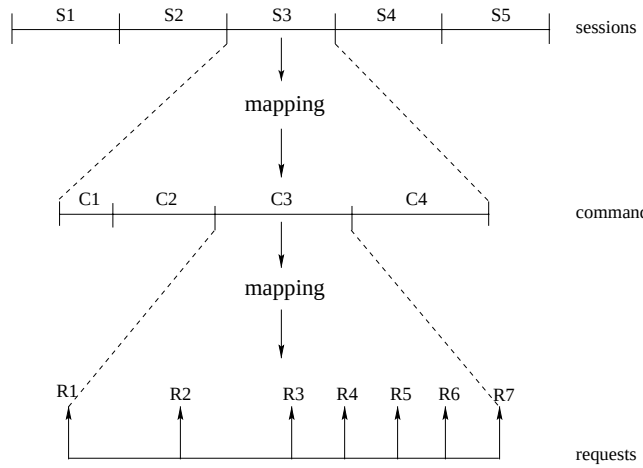


Fig. 2. Three layer hierarchy typical of the networkload.

3 Workload of Client/Server Environments

A client/server environment is typically composed of clients connected to servers through a network. Examples of these environments are distributed file systems, distributed databases, World Wide Web and distributed multimedia systems.

The increasing complexity of the architectures of these systems makes their design and management quite challenging. The introduction of a large variety of advanced services, such as integrated services for mobile users, makes these issues even more crucial. In this framework, performance evaluation and, in particular, workload characterization, are very helpful. Studies related to capacity planning, load balancing, resource allocation and scheduling, and congestion control policies, rely on an accurate characterization of the workload of client/server environments. This characterization is also the basis for the definition of the input parameters of models, aimed, for example, at performance prediction.

In a client/server environment, a client generates requests, which are transmitted through a network, and received by a server. The server processes these requests and sends back the replies to the client. Hence, client/server environments can be seen as hierarchically structured into three layers, namely client, network, and server. Depending on the considered layer, workload consists of the requests, as seen at the client or server layers, or of the packets flowing on the network.

Workload characterization in client/server environments typically relies on the analysis of measurements collected at each layer. The choice of what has to be measured depends on the objective of the study, the workload characteristics to be evaluated, and the availability of tools able to collect the appropriate measures.

At the network layer, measurements, which can span hours and days (see e.g., [22], [37]), or time periods of weeks and months (see e.g., [36]), have been

often collected. Software monitors, such as `tcpdump`, capture the packets flowing on the network, and provide user level control of measurement collection. This control includes filtering on a per host, protocol, or port basis. Information, such as packet arrival times, packet lengths, as well as source and destination host, and port number, are captured. Measurements can also be collected by means of hardware monitors. Tools, such as `sniffers`, capture the packets flowing on the network, and perform some preliminary analyses on them. Timing of the packet arrivals, packet lengths, together with events, such as packet retransmissions, are gathered.

At the client and server layers, the use of accounting routines provides measurements about the processed requests. These measurements may not be enough to fully characterize the behavior of the requests. In order to obtain more detailed information, the availability of ad–hoc tools able to gather the events of interest is required. A simple way to capture these events consists of modifying the source codes of client or server applications by adding custom instrumentation. In the case of Web clients, browsers have been instrumented ([14]). In such a way, the arrival time of each request, the size of the requested file, together with the URL, session, user, and client identifiers, are captured. At the server layer, the instrumentation of the `httpd` daemon provides the CPU and disk demands of each request ([16]). In distributed file systems, kernel instrumentation is used to extract from each remote file system information, such as file lengths, access/modification times ([5]). It should be noted that all these measurement approaches require the availability of source codes.

From the analysis of the measurements collected at the client, network, and server layers, a model of the overall workload of the client/server environment can be obtained. Nevertheless, most of the studies characterize the workload of each layer separately, by using the corresponding measurements.

Early studies at the network layer are based on measurements collected on an Ethernet network (see e.g., [44], [22]). The workload consists of sequences of packets. The parameters typically considered for their characterization are packet interarrival times, packet lengths, error rates, and network utilization. For these parameters, basic statistics, together with their distributions, are computed. Figs. 3 and 4 show the distributions of packet lengths and interarrival times, respectively. As can be seen, the packet lengths are characterized by a bimodal distribution, whereas interarrival time distribution is heavy–tailed. Interarrival times are not statistically independent, that is, with a high probability a packet is followed by a second packet within a deterministic time which depends on the protocol, packet length, and traffic intensity. Hence, Poisson models, historically used for modeling network traffic, are inappropriate to represent packet arrival process.

The network traffic can also be analyzed from a functional view point. In [22], the composition of the traffic has been studied on a per protocol basis. It has been shown that three protocols, namely TCP, NFS, and ND (Network Disk), carry most of the traffic. For each of these protocols, parameters, such as packet interarrival times, are analyzed. From the interarrival time distributions, it has

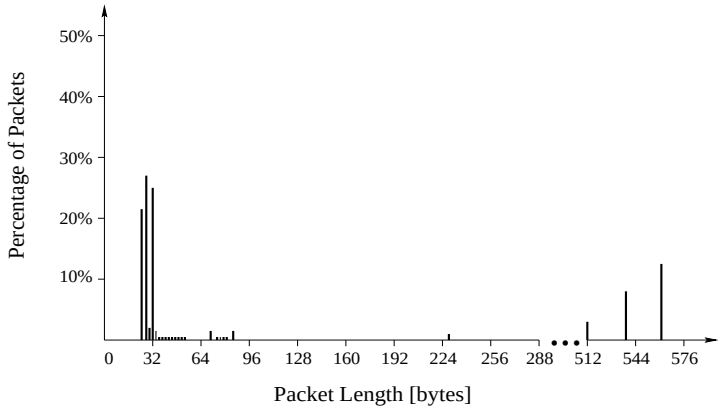


Fig. 3. Distribution of the length of the packets transmitted over the Ethernet [44].

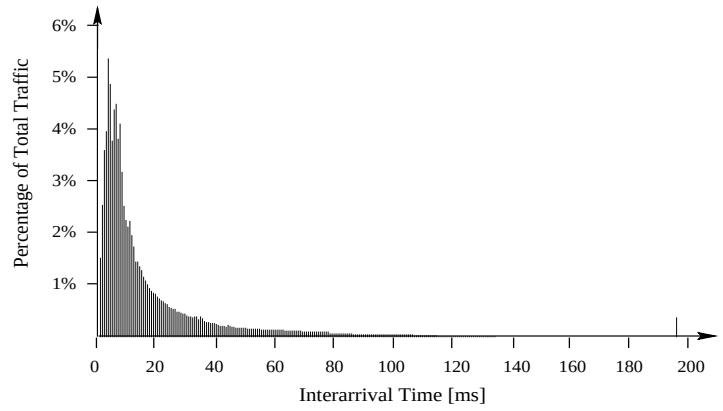


Fig. 4. Distribution of the interarrival times of the packets transmitted over the Ethernet [44].

been discovered, for example, that TCP is "slow" compared to NFS and ND. The 90th percentile of the interarrival time distribution of TCP packets is equal to 52 ms. For ND packets, this percentile is equal to 18 ms.

A functional description of the traffic can also be obtained by recognizing intranetwork and internetwork traffic. From the analysis of source/destination patterns, the distribution of the traffic among servers can be studied. The traffic is typically unevenly distributed and concentrated to and from specialized servers, like gateways, information servers, file servers, print servers.

Recent studies (see e.g., [29], [37]) have introduced the concept of "self–similarity" as a fundamental characteristic of network traffic. This concept ap-

plies to both local and wide area networks. Packet arrivals are characterized by a bursty nature. The traffic measured on the network is seen as the aggregation of bursty traffic generated by independent sources; as the number of traffic sources increases, burstiness of the total traffic increases. The burstiness of the arrivals, already discovered in earlier studies ([44]), is the most evident characteristic of self–similarity. By plotting the packet arrival counts, i.e., the number of packet arrivals per time unit, and changing the time unit, the arrival pattern maintains the same bursty structure on different time scales. Poisson arrivals, on the contrary, become smoother at coarser time scales. This bursty arrival process has been discovered as characterizing the overall and the per protocol traffic. For example, both `telnet` and `ftp` packet arrivals are characterized by a self–similar behavior. Fig. 5 shows the typical behavior of a bursty arrival process. Figs. 5(a) and 5(b) show the count of Ethernet packet arrivals for time units equal to 2.5 and 0.25 seconds, respectively.

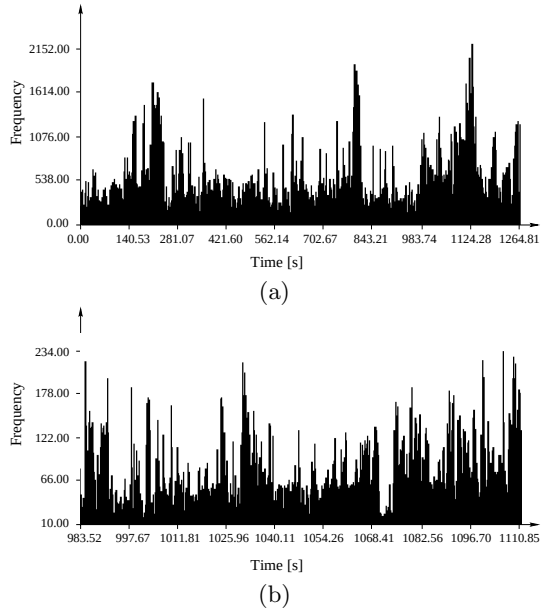


Fig. 5. Frequency of packets for two different time units.

Statistical methods for estimating the degree of self–similarity are the time–domain analysis based on R/S statistic, the variance–time analysis, and the spectral–domain method using periodograms ([29]). Fig. 6 shows the plots of these three statistics for traffic measured during one day on an Ethernet network. All the three methods provide an estimate of the Hurst parameter, which is used for quantifying the degree of self–similarity. In our example, the slopes of the R/S statistic (Fig. 6(a)) and variance curve (Fig. 6(b)) give a value of the Hurst parameter equal to 0.9, which means high degree of self–similarity. The

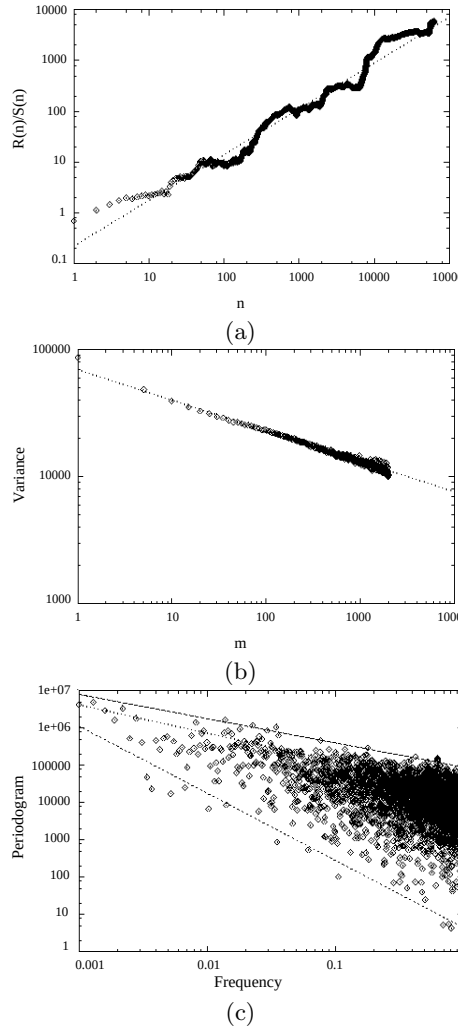


Fig. 6. R/S statistic (a), variance (b), and periodogram (c) for the Ethernet traffic.

spectral–domain method (Fig. 6(c)) is less accurate, and it only confirms what provided by the other two methods.

The phenomenon of self–similarity has also been discovered as characterizing the network traffic associated with the requests issued by Web clients and with replies of Web servers ([31]). In particular, the distribution of the length of the replies is heavy tailed, i.e., the replies are large with non negligible probability, whereas the length of the requests has a bimodal distribution.

At the client and the server layers, the workload consists of the sequences of the requests generated by clients and processed by servers. At the client layer,

requests are described by means of parameters, such as the arrival time, and the size of the requested file. In Web environments, these parameters are characterized by heavy–tailed distributions (see e.g., [15], [14]). In the case of file sizes, the tail of their distribution is due to multimedia applications. Even though the distribution of text files is itself heavy–tailed, the combined use of files carrying text, image, audio, and video, has the effect to increase the length of the tail.

The analysis of the size of a file and of the number of times it is accessed provides a characterization of the user behavior. An inverse correlation has been discovered between these parameters, that is, users tend to access small files.

Workload invariants are identified from measurements on Web servers. These invariants refer to characteristics that apply across different servers ([4]). Invariants are the percentage of accesses to valid files, file types, mean transfer size, percentage of requests to different files, file size distribution, and parameters related to file referencing behavior, e.g., frequency of reference, inter-reference times, percentage of remote requests. A non–uniform referencing behavior characterizes the servers. Fig. 7 shows the cumulative frequency of file accesses for six Web servers. As can be seen, there are very few files responsible for most of the requests received by each server.

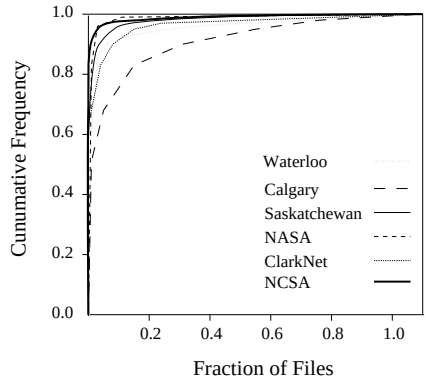


Fig. 7. Cumulative frequency of file accesses for six Web servers [4].

As already pointed out, a model of the overall workload of a client/server environment has to be based on the analysis of measurements collected at network, client, and server layers. These models are typically used as input for analytic and simulation models aimed at predicting the performance of the systems.

In [39], a generative model, which reflects the hierarchical nature of the client/server workload, is built. The data collected at different layers (i.e., session, command, request) is analyzed independently to estimate its characterizing parameters and to reduce it by means of clustering. Further analyses provide estimates of the mapping between layers. A networkload model is then obtained and used as input to simulation model of a single server multiple client network.

In [16], the workload of internet and intranet Web servers is characterized with the aim of deriving the input parameters of a capacity planning model of the Web server. The requests issued by the clients are subdivided into three classes, according to the types of file accessed on the server (i.e., text, image, CGI). Each class of requests is described by its service demands, e.g., CPU time and disk demand at the server, and network delay. A layered queueing network model of the Web server is built for predicting the client and server response times.

In [13], measurements collected at the client and server layers are used in combination to drive the scheduling policies of a distributed Web server. A number of workload parameters are derived from these measurements in order to balance the load among servers.

A generative workload model of a distributed file server is presented in [7]. The workload is described by the frequency distribution of file server requests, the request interarrival time distribution, the file referencing behavior, and the distribution of sizes of read and write requests. The workload model generates requests to the server by sampling these distributions.

Train models, first introduced in [26] to model packet arrivals in a token ring local area network, have been recently used to reproduce self–similarity of ATM, Ethernet, and Web traffic (see e.g., [27], [49], [14]). A train is seen as a sequence of "ON" and "OFF" periods. During ON periods, packets arrive at regular intervals, whereas during OFF periods there are no packet arrivals (see Fig. 8). ON and OFF periods are characterized by heavy–tailed distributions, and are modeled by means of Pareto distributions. The overall traffic of the network can then be generated by superposing sources of traffic, each represented by a train model. Other models, such as fractional Gaussian noises ([33]) and fractional ARIMA processes ([6]), can be used for the generation of self–similar traffic.

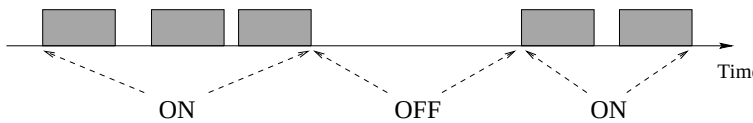


Fig. 8. ON/OFF model.

It should be noted that all these case studies enforce the importance of an accurate characterization of the workload of client/server environments, whose requests are characterized by different performance and quality of service requirements. Indeed, the more detailed the workload description is, the better will be the performance predictions.

4 Workload of Parallel Systems

The workload of parallel systems is represented by the applications being processed. The driving factor in developing parallel applications is to get better performance, i.e., to solve a given problem in a shorter time or to be able to solve bigger problems. Workload characterization focuses on the performance issues which have to be addressed to meet such objectives. Tuning, performance debugging and diagnosis require an accurate characterization of parallel applications. A detailed description of parallel applications is also required by performance prediction studies aimed, for example, at the design and evaluation of scheduling policies.

A parallel application is a collection of interrelated tasks that interact through a predetermined flow of control. Each task consists of a functional block (i.e., a sequence of statements) which is executed on a given processor; in turn, different tasks can be allocated to different processors. Data and functional dependencies existing among the various tasks are translated into communications and synchronizations activities, which can be based on explicit message passing or on shared memory. A typical representation of parallel applications is based on task graphs whose nodes correspond to the functional blocks and whose arcs correspond to the dependencies existing among them. Fig. 9 shows an example of a task graph.

From the task graph, parameters and metrics are derived in order to obtain qualitative and quantitative descriptions of the behavior of parallel applications.

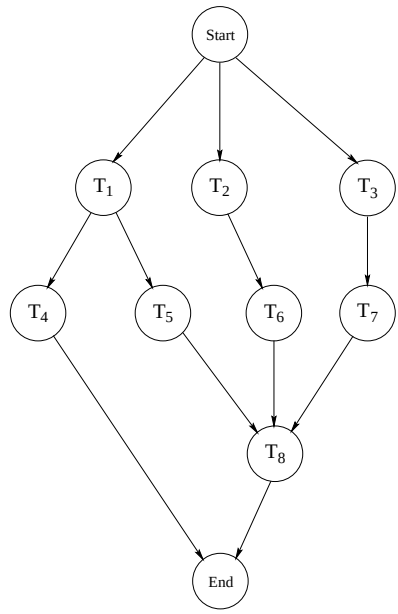


Fig. 9. Example of a task graph.

Early studies (see e.g., [42], [32]) focus on the analysis of these graphs with the aim of describing the inherent parallelism of the applications.

Parameters, like the maximum cut, the depth, the average parallelism, and the sequential fraction, can be directly derived from the task graph. The maximum cut, that is, the maximum number of arcs taken over all possible cuts from the initial to the final node, provides the maximum theoretical degree of parallelism. The depth, that is, the longest path between the initial and final node, is directly related to the overall execution time. The average parallelism is defined as the average number of "active" tasks during the execution of the application. The sequential fraction is the fraction of the overall execution time which cannot benefit from parallel execution ([3]).

From the task graph, the overall execution time of an application has to be estimated by computing the execution time of each task and considering the data and functional dependencies existing among tasks. The execution time of each task is itself estimated by considering the number of instructions executed, the number of clock cycles per instruction, and the processor clock cycle time.

All these parameters provide preliminary insights into the parallelism that can be exploited by an application. The dynamic aspects of the application are not captured. Metrics, like the parallelism profile and the application shape, have to be used to investigate such aspects. The parallelism profiles and the application shape can refer to estimated or actual execution of the application. The parallelism profile plots, as a function of the execution time, the number of active tasks. The application shape is the fraction of execution time in which a given number of tasks is active. Examples of these metrics are shown in Fig. 10.

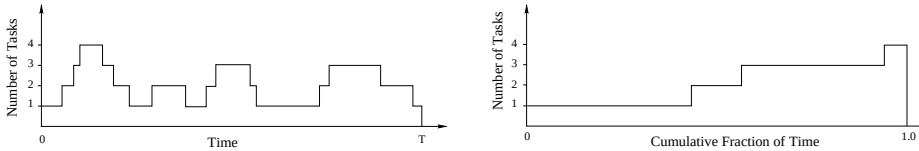


Fig. 10. Parallelism profile (a) and shape (b) of a parallel application [42].

More details about the behavior of parallel applications are obtained by incorporating into the parallelism profile various types of parallelism delay, such as contention and access delays to shared resources, overhead delays of transferring and initiating the various tasks of the application.

Even though these profiles are based on the analysis of the task graph and on estimates of the achievable performance only, they represent a first attempt to take into account the interactions of the application with the underlying parallel system. However, for most scientific parallel applications building a task graph is infeasible. Moreover, the actual performance is influenced by a large number of aspects related to the algorithms used and to their interactions with the parallel

system. It is then important to gather information from the execution of the applications. Measurement based approaches work for this purpose. Depending on the objective of the study, measurements are collected at various levels of details. The selection of what has to be measured is critical. Indeed, there is a tradeoff between the amount of details to be measured and the perturbations introduced in the behavior of the application being measured.

Measurements are obtained by monitoring the applications. Instrumentation has to be statically or dynamically added to operating system schedulers, communication libraries, or source code of the applications ([25]). Events, associated, for example, with the begin or end of portions of application source code (e.g., subroutines, loops, arbitrary sequences of statements) or with communication statements, are monitored. During the execution of an application, when an event occurs, a time stamp, and the identifiers of the event itself and of the processor where the event occurred, are captured.

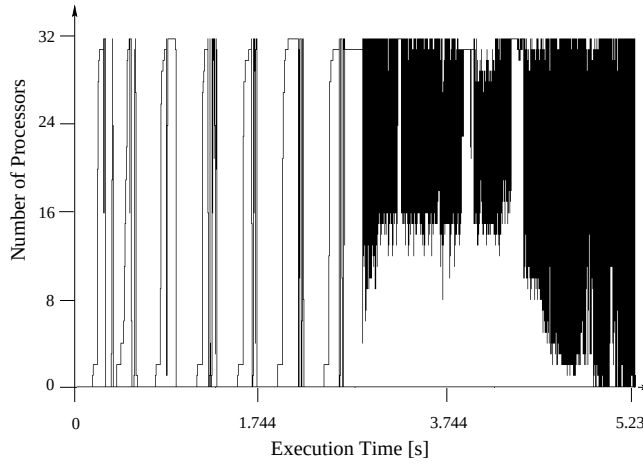
Flexible techniques have to be applied for selecting and presenting the most relevant characteristics of the application from the overwhelming amount of details contained into the measurements. In this framework, visualization techniques are important in that they highlight the behavior of the application by means of various types of graphical representation (see e.g., [24]).

A parallel application can be analyzed under different perspectives and at different levels of abstraction. Parameters and metrics describing its static and dynamic behavior are extracted from the measurements. Preliminary information about the behavior of the whole application can be provided by timing parameters, such as execution, computation, communication, and I/O times, and volume parameters, such as number of floating point operations, communications, and I/O operations. These parameters summarize the behavior of the application by showing the tradeoff between actual computation, communication and synchronization activities, and I/O requirements.

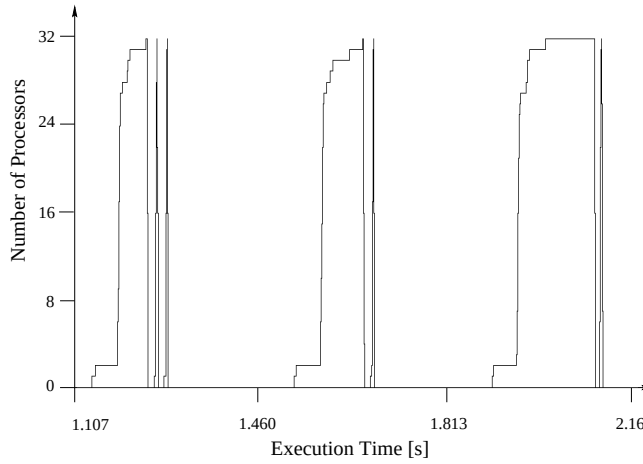
The dynamic behavior of an application is described by means of profiles, which express, as a function of the execution time, the number of processors involved simultaneously in a specific activity. Execution profiles describe the overall behavior of the applications.

In the case of applications which rely on communication libraries, such as MPI and PVM, the characterization of communication activities is particularly critical because of the large variety of available protocols and buffering policies. Communication profiles reflect the overall behavior of the communications. Moreover, the profiles can be specialized on a per protocol basis. Fig. 11(a) shows an example of a communication profile for an application executed with 32 processors. The profile corresponding to the MPI Allreduce protocol is shown in Fig. 11(b).

The analysis of the parallel applications as a function of the number of allocated processors, that is, the scalability of the applications, can be addressed by performance metrics, derived from timing parameters. The speedup is a figure of merit of the exploitation of available parallelism, in that, it is a measure of how much the execution time decreases with an increase in the number of allocated



(a)



(b)

Fig. 11. Communication profile (a) and MPI Allreduce profile (b).

processors. Fig. 12 shows a speedup curve of an application executed with a number of processors ranging from 1 up to 64.

Note that the increase of the number of allocated processors does not always lead to performance improvements. As can be seen, there is a degradation of the performance when the number of processors is increased from 32 to 64. The benefit from allocating additional processors does not compensate the costs due to the increased synchronization and communication activities.

The effective exploitation of the allocated processors can also be assessed by performance metrics, like efficiency and efficacy.

There is a number of studies, e.g., design of scheduling policies, performance prediction, tuning, performance debugging and diagnosis, which rely on a detailed characterization of parallel applications. For example, when dealing with the design and the evaluation of scheduling policies, a parallel application is described in terms of number of processors it requires and the time it takes to be executed (see e.g., [20], [43]). Moreover, the arrival process of the applications has to be specified by appropriate distributions, such as Poisson distributions and experimental distributions derived by fitting actual arrivals ([17]).

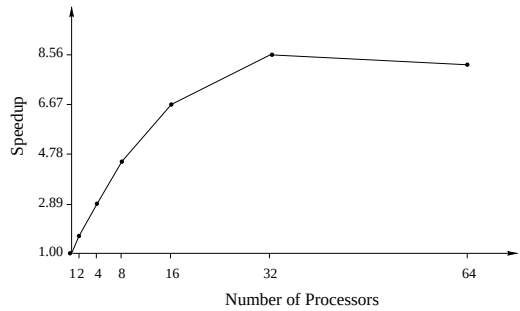


Fig. 12. Speedup curve as a function of the number of allocated processors.

Other performance studies, dealing with performance debugging and diagnosis, focus on the analysis of the various portions of application source code, e.g., loops, subroutines, arbitrary code segments. Profiling techniques provide for each portion of the code the execution time together with the number of times it has been executed. In such a way, as in the case of sequential applications, the "heaviest" portions of the code, where the application spends its time, are identified.

Moreover, the behavior of an application can be described under a multiplicity of views, each addressing a specific aspect of its execution ([28]). For example, processor and data parallelism views can be obtained. These views describe the behavior of the individual processors, with respect to the data it has to process, and the interactions among the processors, with respect to their communication and synchronization activities.

The analysis of the behavior of the individual processors is particularly useful for data parallel applications where the parallelism is basically exploited by distributing the data among the allocated processors. For example, processor views identify the presence of unevenly distributed data. In such a case, the processors are characterized by "irregular" behavior, that is, the load of the various processors becomes unbalanced and synchronization delays arise.

Another approach toward a detailed description of a parallel application is based on the identification of phases, used to represent the execution behavior of the application. The phases refer to the algorithms used or to specific activities,

such as computation, communication, and I/O. An application is then seen as a sequence of phases.

In [12], the execution profile of an application is analyzed with the aim of identifying the phases it consists of. The profile is seen as a sequence of alternating periods of roughly uniform processor utilization separated by periods of sudden transition in processor utilization. These phases are analyzed with respect to their speedup behavior in order to gain insights into the performance and the scalability of the application. Fig. 13(a) shows an example of an execution profile. As can be seen, there are periods characterized by high variance in the number of utilized processors. The use of smoothing techniques reduces this variance and eases the identification of the phases. Fig. 13(b) shows the execution profile smoothed by least squares.

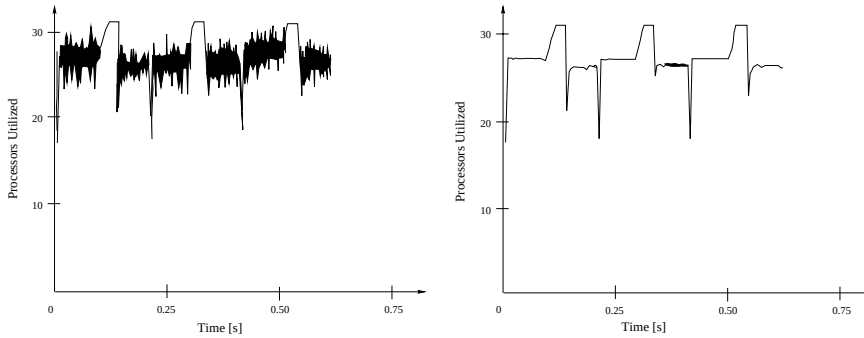


Fig. 13. Execution profile (a) and profile smoothed by least squares (b) [12].

In [48], the characterization of the phases of computational fluid dynamics applications, which solve partial differential equations, yields the identification of tuning methodologies based on resource utilization profiles as well as the evaluation of the performance of heterogeneous metacomputing environments. This characterization relies on a top–down approach. Measurements collected at the system level, together with a domain specific knowledge of this class of applications, are used to characterize the phases. The application is seen as alternating computation and communication phases. This alternation is described by means of interarrival times between successive communication phases, length of computation and communication phases. Each of these parameters is specified by an appropriate probability distribution.

The phases identified in this study are typical of explicit message passing applications, whose parallelism is expressed by means of communication statements. In the case of parallel applications which use high level languages, such as High Performance Fortran (HPF), the parallelism is expressed by means of directives provided by the language itself. The compiler will take advantage of these directives to parallelize the applications. The performance of these applications

is influenced by the parallelization strategies adopted by the compiler, whereas the performance of explicit message passing applications is mainly influenced by communication mechanisms.

Explicit message passing applications require a detailed characterization of the communication activities (see e.g., [34], [47]). The characterization of HPF applications has to focus on the costs of the parallelization strategies (see e.g., [9]). These costs are seen as overhead, which results into an increase of the execution time of the application. The execution time can be broken down according to the various activities required by the parallelization. The evaluation of these costs is useful for both application and compiler developers. The impact of HPF directives on the performance of the application and the efficiency of the parallelization strategies adopted by the compiler are assessed. Fig. 14 shows the profiling of the `INDEPENDENT` loops of an HPF application. For each loop, the figure also presents the breakdown of the execution time with respect to the costs associated with the distribution of work and data among the allocated processors.

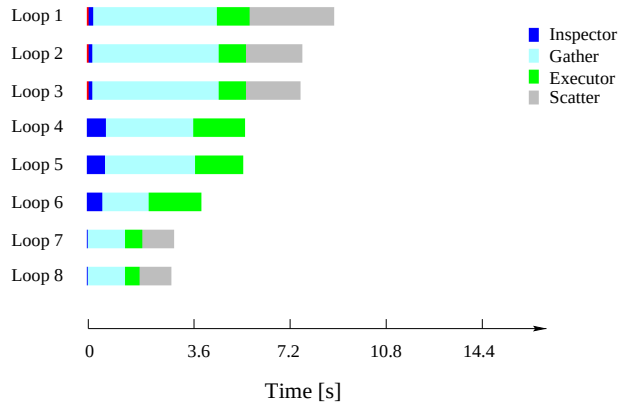


Fig. 14. Breakdown of the execution time of HPF `INDEPENDENT` loops.

Parallel scientific applications, typically described as computation intensive, are becoming more I/O intensive, due to the large volume of data to be manipulated. The efficiency and the scalability of an application are influenced by its I/O requirements. Moreover, these requirements influence the design of high performance I/O systems.

Recent studies (see e.g., [35], [45]), have addressed the characterization of I/O requirements of parallel applications. These studies analyze a set of applications, whose access patterns are representative of the behavior of most scientific applications. The characterization focuses on the behavior of I/O requests, such as seeks, reads, and writes. Each of these I/O requests is described by its count and its duration. The analysis of the temporal and spatial patterns of I/O re-

quests shows the burstiness of the accesses and their non sequential behavior, as well as the presence of interleaved and strided patterns. These patterns help in identifying phases in the I/O activities of an application. An application typically starts by reading its data set, then, it might use temporary files, as in the case of out of core computations. Finally, the application writes back the results of its execution.

The distributions of the size of various types of request describe the variability of the volume of the data being accessed. Fig. 15 shows the cumulative distributions of the read sizes for three scientific applications. As can be seen, these distributions are very different.

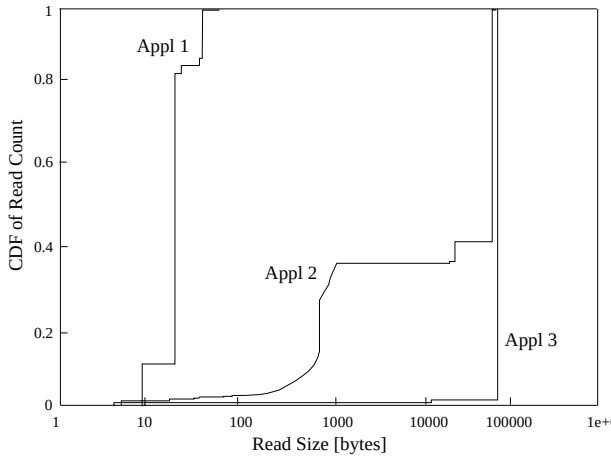


Fig. 15. Cumulative distributions of read sizes of three scientific applications [45].

In [40], a model of parallel applications, used to examine the impact of the I/O on their behavior and analyze resource scheduling policies, is formulated. The model captures the I/O and computation characteristics of the applications. An application is seen as a sequence of phases, consisting of a single burst of computation followed by a single burst of I/O. A generalization of the speedup with respect to both processor and disk resources is also derived.

5 Future Directions

The increased complexity of computer architectures and of their operating environments and the advent of new frontiers of computing enforce the use of workload characterization as the basis for performance studies. Design and tuning of new emerging systems, such as mail servers, name servers, Java virtual machines, mobile systems, require accurate studies of their workloads. The same

applies to the design of software components, such as graphical user interfaces, teleconferencing applications, parallelizing compilers. For example, the performance of applications with graphical user interfaces is the reflection of user perception. Hence, the characterization of these applications has to rely on low–level measurements of the user generated events, such as keystrokes and mouse clicks, which represent the load of the systems.

New measurement and methodological approaches have to be introduced in order to cope with all these performance requirements which have also to be coupled with quality of service requirements. Moreover, since workload characterization, like performance evaluation, is still considered more as an art than as a science, the availability of integrated environments and tools able to address the performance issues encountered in the various application domains will help in mastering this art. For example, the design of efficient policies for managing the hardware and software resources of the systems will benefit from an automated process for collecting, analyzing, and modeling workload measurements. The development of tools able to automatically identify the bottlenecks and the sources of performance degradation, and to provide hints about possible solutions will also be very useful.

Acknowledgements

This work was supported in part by the European Commission under the ESPRIT IV Working Group APART, the Italian Space Agency (ASI) and the University of Pavia under the FAR program.

References

1. A.K. Agrawala and J.M. Mohr. A Markovian Model of a Job. In *Proc. CPEUG*, pages 119–126, 1978.
2. A.K. Agrawala, J.M. Mohr, and R.M. Bryant. An Approach to the Workload Characterization Problem. *Computer*, pages 18–32, 1976.
3. G.M. Amdahl. Validity of the Single–processor Approach to Achieving Large Scale Computing Capabilities. In *Proc. AFIPS Conf.*, volume 30, pages 483–485, 1967.
4. M.F. Arlitt and C.L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proc. ACM SIGMETRICS Conf.*, pages 126–137, 1996.
5. M. Baker, J. Hartman, M. Kupfer, K. Shirriff, and J. Ousterhout. Measurements of a Distributed File System. In *Proc. ACM Symposium on Operating Systems Principles*, pages 198–212, 1991.
6. J. Beran. Statistical Methods for Data with Long–range Dependence. *Statistical Science*, 7(4):404–427, 1992.
7. R.R. Bodnarchuk and R.B. Bunt. A Synthetic Workload Model for a Distributed System File Server. In *Proc. ACM SIGMETRICS Conf.*, pages 50–59, 1991.
8. M. Calzarossa, G. Haring, and G. Serazzi. Workload Modeling for Computer Networks. In U. Kastens and F.J. Ramming, editors, *Architekture und Betrieb von Rechensystemen*, pages 324–339. Springer–Verlag, 1988.
9. M. Calzarossa, L. Massari, and D. Tessera. Performance Issues of an HPF–like compiler. *Future Generation Computer Systems*, 1999.

10. M. Calzarossa and G. Serazzi. A Characterization of the Variation in Time of Workload Arrival Patterns. *IEEE Trans. on Computers*, C-34(2):156–162, 1985.
11. M. Calzarossa and G. Serazzi. Workload Characterization: a Survey. *Proc. of the IEEE*, 8(81):1136–1150, 1993.
12. B.M. Carlson, T.D. Wagner, L.W. Dowdy, and P.H. Worley. Speedup Properties of Phases in the Execution Profile of Distributed Parallel Programs. In R. Pooley and J. Hillston, editors, *Modelling Techniques and Tools for Computer Performance Evaluation*, pages 83–95. Antony Rowe, 1992.
13. M. Colajanni, P.S. Yu, and D.M. Dias. Analysis of Task Assignment Policies in Scalable Distributed Web-server System. *IEEE Trans. on Parallel and Distributed Systems*, 9(6), 1998.
14. M.E. Crovella and A. Bestavros. Self–similarity in World Wide Web Traffic: Evidence and Possible Causes. In *Proc. ACM SIGMETRICS Conf.*, pages 160–169, 1996.
15. C.R. Cunha, A. Bestavros, and M.E. Crovella. Characteristics of WWW Client–based Traces. Technical Report BU-CS-95-010, Computer Science Dept., Boston University, 1995.
16. J. Dilley, R. Friedrich, T. Jin, and J. Rolia. Web Server Performance Measurement and Modeling Techniques. *Performance Evaluation*, 33(1):5–26, 1998.
17. D.G. Feitelson and L. Rudolph. Metrics and Benchmarking for Parallel Job Scheduling. In D.G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 1998.
18. D. Ferrari. Workload Characterization and Selection in Computer Performance Measurement. *Computer*, 5(4):18–24, 1972.
19. D. Ferrari. On the Foundations of Artificial Workload Design. In *Proc. ACM SIGMETRICS Conf.*, pages 8–14, 1984.
20. D. Ghosal, G. Serazzi, and S.K. Tripathi. The Processor Working Set and its Use in Scheduling Multiprocessor Systems. *IEEE Trans. on Software Engineering*, SE-17(5):443–453, 1991.
21. S.D. Gribble, G.S. Manku, D. Rosselli, E.A. Brewer, T.J. Gibson, and E.L. Miller. Self–similarity in File Systems. In *Proc. ACM SIGMETRICS Conf.*, pages 141–150, 1998.
22. R. Gusella. A Measurement Study of Diskless Workstation Traffic on an Ethernet. *IEEE Trans. on Communications*, COM-38(9):1557–1568, 1990.
23. G. Haring. On Stochastic Models of Interactive Workloads. In A.K. Agrawala and S.K. Tripathi, editors, *PERFORMANCE '83*, pages 133–152. North–Holland, 1983.
24. M.T. Heath, A.D. Malony, and D.T. Rover. Parallel Performance Visualization: from Practice to Theory. *IEEE Parallel and Distributed Technology*, 3(4):44–60, 1995.
25. R. Hofmann, R. Klar, B. Mohr, A. Quick, and M. Siegle. Distributed Performance Monitoring: Methods, Tools, and Applications. *IEEE Trans. on Parallel and Distributed Systems*, 5(6):585–598, 1994.
26. R. Jain and S.A. Routhier. Packet Trains - Measurements and a New Model for Computer Network Traffic. *IEEE Journal on Selected Areas in Communications*, SAC-4(6):986–995, 1986.
27. J.L. Jerkins and J.L. Wang. Cell–level Measurement Analysis of Individual ATM Connections. *Workshop on Workload Characterization in High–Performance Computing Environments*, 1998. http://sokrates.ani.univie.ac.at/~gabi/wlc.mascots.

28. T. Le Blanc, J. Mellor-Crummey, and R. Fowler. Analyzing Parallel Program Executions Using Multiple Views. *Journal of Parallel and Distributed Computing*, 9(2):203–217, 1990.
29. W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the Self–similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Trans. on Networking*, 2(1):1–15, 1994.
30. P.A. Lewis and G.S. Shedler. Statistical Analysis of Non–stationary Series of Events in a Data Base System. *IBM Journal on Research and Development*, 20:465–482, 1976.
31. B.A. Mah. An Empirical Model of HTTP Network Traffic. In *Proc. IEEE InfoCom '97*, 1997.
32. S. Majumdar, D. Eager, and R. Bunt. Characterization of Programs for Scheduling in Multiprogrammed Parallel Systems. *Performance Evaluation*, 13(2):109–130, 1991.
33. B.B. Mandelbrot and J.W. Van Ness. Fractional Brownian Motions, Fractional Noises and Applications. *SIAM Review*, 10:422–437, 1968.
34. A. Merlo and P.H. Worley. Analyzing PICL trace data with MEDEA. In G. Haring and G. Kotsis, editors, *Computer Performance Evaluation*, volume 794 of *Lecture Notes in Computer Science*, pages 445–464. Springer–Verlag, 1994.
35. N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Ellis, and M. Best. File–access Characteristics of Parallel Scientific Workloads. *IEEE Trans. on Parallel and Distributed Systems*, 7(10):1075–1088, 1996.
36. V. Paxson. Growth Trends in Wide–area TCP Connections. *IEEE Network*, 8(4):8–17, 1994.
37. V. Paxson and S. Floyd. Wide–area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Trans. on Networking*, 3(3):226–244, 1995.
38. S.V. Raghavan, P.J. Joseph, and G. Haring. Workload Models for Multiwindow Distributed Environments. In H. Beilner and F. Bause, editors, *Quantitative Evaluation of Computing and Communication Systems*, pages 314–326. Springer, 1995.
39. S.V. Raghavan, D. Vasukiammaiyar, and G. Haring. Generative Networkload Models for a Single Server Environment. In *Proc. ACM SIGMETRICS Conf.*, pages 118–127, 1994.
40. E. Rosti, G. Serazzi, E. Smirni, and M. Squillante. The Impact of I/O on Program Behavior and Parallel Scheduling. In *Proc. ACM SIGMETRICS Conf.*, pages 56–65, 1998.
41. G. Serazzi. A Functional and Resource–oriented Procedure for Workload Modeling. In F.J. Kylstra, editor, *PERFORMANCE '81*, pages 345–361. North–Holland, 1981.
42. K. Sevcik. Characterization of Parallelism in Applications and Their Use in Scheduling. In *Proc. ACM SIGMETRICS Conf.*, pages 171–180, 1989.
43. K.C. Sevcik. Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems. *Performance Evaluation*, 19:107–140, 1994.
44. J.F. Shoch and J.A. Hupp. Measured Performance of an Ethernet Local Network. *Communications of the ACM*, 23(12):711–721, 1980.
45. E. Smirni and D. Reed. Lessons from characterizing the input/output behavior of parallel scientific applications. *Performance Evaluation*, 33(1):27–44, 1998.
46. K. Sreenivasan and A.J. Kleinman. On the Construction of a Representative Synthetic Workload. *Communications of the ACM*, 17(3):127–133, 1974.
47. D. Tessera, M. Calzarossa, and A. Malagoli. Performance Analysis of a Parallel Hydrodynamic Application. In A. Tentner, editor, *High Performance Computing*, pages 33–38. SCS Press, 1998.

48. A. Waheed and J. Yan. Workload Characterization of CFD Applications Using Partial Differential Equation Solvers. *Workshop on Workload Characterization in High–Performance Computing Environments*, 1998. http://sokrates.ani.univie.ac.at/~gabi/wlc.mascots.
49. W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson. Self–similarity Through High–variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level. In *Proc. ACM SIGCOMM Conf.*, pages 100–113, 1995.
50. M. Zhou and A.J. Smith. Tracing Windows95. Technical Report, Computer Science Division, UC Berkeley, November 1998.