Quantifying the Cluster Speedup Behavior in the Realm of Internode Communication

Dominique A. Heger & Greg Simco

Proceeding of the IEEE Southeast Conference, Fort Lauderdale. April2005

Abstract

Over the past decade, there has been an increased interest in designing and implementing high-performance cluster systems by utilizing commodity components that are connected through standard network technology. Networked clusters of computers are commonly used to either process multiple sequential jobs concurrently, or to execute complex scientific and commercial parallel applications that are based on a message passing paradigm. In some areas, these clusters represent a cost-effective alternative to the more expensive supercomputers that are in use today. However, workstation clusters normally lack the high-powered interconnect fabric, as well as the more optimized protocol system architecture that is an integral part of commercial supercomputer systems. This study quantifies the speedup of parallel applications in a cluster environment in an analytical model that takes parallel processing, internode communication, and I/O latency into consideration. The model illustrates the (relative) impact that CPU and I/O parallelism has on the speedup behavior based on the 3 different inter-node communication scenarios: broadcast, nearest-neighbor, and request-response. This work revealed that it is important to consider the performance impact that the application has with regard to features such as remote memory access, completion notification, and address translation issues.

1 Introduction

A cluster can be described as a collection of independent compute and I/O nodes that are interconnected through a network. These networks of workstations (which represent a virtual, distributed-memory parallel machine) are used to process compute intensive parallel applications. A cluster represents the natural increment from a symmetric multiprocessor (SMP) system, and hence fills the gap between a single node system and a supercomputer. As in the case of commercial Massively Parallel Processing (MPP) systems, a cluster is highly scalable, as nodes can easily be added, which increases the potential processing power. The argument made is that two factors limit the effectiveness of cluster systems when compared to MPP supercomputers. First, a cluster's communication performance is normally inferior to supercomputers. Emerging technologies such as Myrinet and Infiniband [4], as well as enhancements in the protocol's systems architecture are focused on closing the gap. Second, scientific applications in general process large quantities of data that not only have to be exchanged among the CPU's, but also have to be shared among the available I/O devices. Thus, a potential drawback of a cluster system is that the applications have to be partitioned to run on multiple nodes. This emphasizes communication issues in that these partitioned programs have to cooperate and share resources. The most important resource (to alleviate the shared image issue) is the file system. In the absence of a sound cluster file system, individual components of a partitioned program have to share the cluster storage on an ad-hoc basis. Such an approach generally complicates the programming aspect, limits performance, and compromises the reliability of the cluster environment. Parallel file systems such as IBM's General Parallel File System (GPFS), RedHat's GFS, or Cluster File System's Luster offer a scalable I/O facility [10],[19]. This study introduces an analytical model to evaluate the speedup behavior of parallel applications, focusing on different communication scenarios. This study extends the research conducted by Koenigs [6], Gennaro [20], and Cremonesi [21]. The speedup is expressed as a function of the number of nodes and available disks. Section 2 discusses the speedup and the scaleup functions, respectively, and elaborates on some of the contemporary cluster design issues. Section 3 outlines the reference model being used to discuss the speedup equation, and introduces the basic cluster-computing model. Section 4 discusses three distinct application communication patterns: broadcast, nearest-neighbor, and request-response, and outlines the different performance behaviors utilizing an empirical analysis on an SMP system. Section 5 elaborates (by utilizing the same communication patterns as discussed in Section 4) on the impact that thread synchronization and application communication issues have on the speedup behavior. Section 6 discusses the results of the study and concludes with some remarks on the project's accomplishments.

2 Speedup, Scaleup, and Efficiency

From a software perspective, parallelism within a program may occur at many different levels. A coarse-grain design indicates that the units of parallelism are of a significant length of duration, and refers to parallelism implemented at a rather high level. A fine-grain design indicates that the units of parallelism are of relatively short duration, and is used to describe loop-level parallelization or data parallelism [6]][8]. The characteristics of a parallel application are measured in terms of speedup and scaleup. The speedup function summarizes the efficiency of a parallel algorithm. For a fixed data set, the speedup function captures the decrease in execution time that can be obtained by increasing the number of available processors. The scaleup function captures how well the parallel algorithm handles larger data sets when additional processors become available. A factor that effects parallel applications and operating systems alike is known as Amdahl's law [1]. What Amdahl noted in his research was that there is always some irreducible part of an application or algorithm that can not be decomposed any further to run in parallel. A basic mathematical formulation of Amdahl's law is that the total execution time *t(1)* of any program can be divided into one part that can be processed in parallel and another part that has to be executed sequentially. By defining τ as the serial time, the parallel time can be expressed as $t(1)(1 - \tau)$. By defining p as the number of CPUs, the speedup function can be expressed as:

$$s(p) = \frac{t(1)}{t(p)} = \frac{t(1)}{\tau \cdot t(1) + \frac{(1-\tau)\cdot t(1)}{p}}$$

$$s(p) = \frac{p}{p\cdot\tau + 1 - \tau} = \frac{p}{1 + \tau\cdot(p-1)} \qquad (1)$$

$$s(p) = \frac{1}{\frac{1+\tau\cdot(p-1)}{p}} \qquad \lim_{p\to\infty} \frac{1}{\tau}$$

In Equation 1, τ basically depicts the serialization factor in Amdahl's law. The speedup calculation is straightforward, but reveals some very interesting implications. Assuming that 4% of an algorithm or application is irreducibly serial, Amdahl's law implies that on a 16-CPU system, the obtained speedup is equal to 10. Further, Amdahl's law determines that no matter how many processors are configured on the system, with 4% serialization in an algorithm, the maximum possible speedup equals to 25. As the number of processors increases, the parallel portion approaches zero, and the total execution time approaches the serial part (the 4%) of the equation. The ramification is that even if a lot of processors are configured in a parallel system, the speedup only affects the parallel part of the equation. A variant of the speedup function is known as the efficiency equation. Efficiency is defined as the speedup divided by the number of processors available on the system. In other words, the efficiency of an algorithm can be expressed as the fraction of the total potential processing power that is actually being used. An algorithm with a linear speedup is defined as being 100% efficient [2] [3].

2.1 Cluster Computing Model

This study considers an architectural model consisting of a fixed number of compute nodes and a fixed number of disks that are either connected to the compute nodes or are connected to dedicated I/O nodes. The nodes are interconnected through a switching fabric (some form of communication network). The switching fabric may consist of a Storage Area Network (SAN), a Myrinet, an Infiniband, or a Gigabit Ethernet based solution. From an application analysis perspective, it is imperative to define the time distribution with respect to the potential concurrency among CPU computation, interprocessor communication, and I/O processing scenarios. Studies focused on analyzing parallel scientific applications have shown a rather regular and cyclic workload processing behavior [9] [11]. Therefore, from an application processing perspective, this study characterizes a parallel scientific application as executing in two logically disjunct phases. Phase 1 is defined as the CPU phase, where cyclic CPU-computations alternate with inter-node communication calls to initiate a data exchange. Phase 2 is labeled as the I/O phase, where disk requests are processed concurrently for purposes such as checkpoint restart or to visualize the computed test results. This decomposition-based approach correlates with research conducted by Koenigs [6] and Wu [11]. These studies have shown that the workload properties of many parallel scientific applications can be decomposed into disjoint intervals, each consisting of a CPU phase followed by an I/O phase. The execution behavior of the entire application can therefore be defined as a sequence of processing cycles.

3 Generic Scaleup Model

The focus in this study is on cluster systems that consist of α identical cluster processing nodes and β disks that are either connected to the processing nodes or to one or more I/O nodes. While executing the scientific parallel application, each the CPU and the I/O phase consist of a sequential and a parallel processing component. In this study the mean number of processing cycles that characterizes the parallel scientific application is defined as δ. Further, *seq(cpu)* and *seq(io)* define the mean sequential CPU and I/O time components, whereas *pa(cpu)* and *pa(io)* represent the mean parallel CPU and I/O time components, respectively. The approach is to decompose the parallel application part (as outlined in Amdahl's Law in Equation 1) into a parallel CPU phase and a parallel I/O phase.

As discussed by Gennaro [20], the reference model utilized in this study can be expressed as a queuing network that corresponds to the program execution behavior on a single cluster node. The assumption is that the baseline system is configured with a single CPU and a single disk. Such a system loosely resembles the Google cluster, which consists of 6,000 nodes (each configured with 1 CPU) and 12,000 disks [4]. In the model presented here, a single task enters the system and circulates through the queuing network. In each cycle the task has a probability p ($p = 1/\delta$) to exit the system. The expected execution time $\varepsilon(1)$ for a circulating task that is running on a single node can therefore be defined as:

$$\varepsilon_1 = \delta \cdot \left(seq_{cpu} + pa_{cpu} + seq_{io} + pa_{io}\right) \quad (2)$$

Equation 2 is used as a reference model to evaluate the speedup behavior of some parallel scientific applications. The average execution time of a parallel program on a cluster system with ϕ CPU's (α nodes * φ CPU's per node) and β disks can now be expressed as:

$$\varepsilon(\phi,\beta) = \delta \cdot \left(seq_{cpu} + \phi \cdot \varepsilon_{cpu} + seq_{io} + \beta \cdot \varepsilon_{io}\right) \quad (3)$$

Based on Equation 2 and Equation 3, it is feasible to formulate a first order speedup relation behavior that represents Equation 1 (an actual generalization of Amdahl's speedup law) and that can be expressed as:

$$spdp(\phi,\beta) = \frac{\delta \cdot \left(seq_{cpu} + pa_{cpu} + seq_{io} + pa_{io}\right)}{\delta \cdot \left(seq_{cpu} + \phi \cdot \varepsilon_{cpu} + seq_{io} + \beta \cdot \varepsilon_{io}\right)} \quad (4)$$

The two components $\phi*\varepsilon(cpu)$ and $\beta*\varepsilon(io)$ define the average execution time of the parallel CPU and I/O components of the scientific applications, respectively. The first-order queuing model used in this study assumes that the parallel work (which corresponds to the *pa(cpu)* component of the equation) is uniformly distributed among α nodes. The first-order model assumes that ($\alpha = \phi$), basically stating that every node is configured with a single CPU. Further, the model assumes that the parallel I/O activity *pa(io)* is uniformly distributed among β disks, which basically reflects the standard behavior of a parallel (clustered) file system. Based on these assumptions, the study formulates the average execution time of the parallel scientific application as:

$$\varepsilon(\phi,\beta) = \delta \cdot \left(seq_{cpu} + \frac{pa_{cpu}}{\phi} + seq_{io} + \frac{pa_{io}}{\beta}\right) \quad (5)$$

The model discussed in Equation 5 assumes deterministic and generally identical behavior of the processors and disk drives that are configured in the cluster system. By dividing Equation 2 by Equation 5, it is feasible to express the speedup as a function of the number of CPU's ϕ ($\alpha = \phi$) and the number of disks β [20],[21]. This behavior is further outlined and elaborated in Equation 6 and Equation 7, respectively.

$$spdp_0(\phi,\beta) = \frac{\delta \cdot \left(seq_{cpu} + pa_{cpu} + seq_{io} + pa_{io}\right)}{\delta \cdot \left(seq_{cpu} + \frac{pa_{cpu}}{\phi} + seq_{io} + \frac{pa_{io}}{\beta}\right)} \quad (6)$$

$$spdp_0(\phi,\beta) = \frac{1}{\psi_{cpu} + \frac{\omega_{cpu}}{\phi} + \psi_{io} + \frac{\omega_{io}}{\beta}} \quad (7)$$

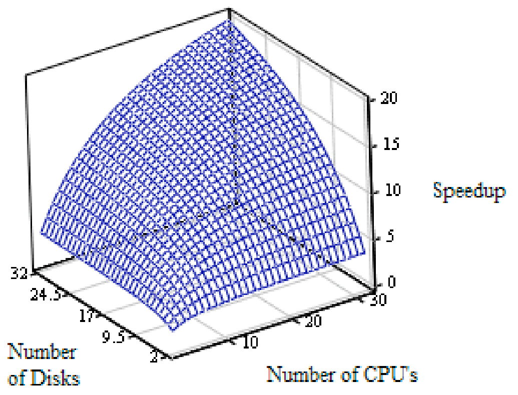
As $q \in \{cpu,io\}$, $\psi(q)$ and $\omega(q)$ can be defined:

$$\psi_q = \frac{seq_q}{seq_{cpu} + pa_{cpu} + seq_{io} + pa_{io}} \quad (8)$$

$$\omega_q = \frac{pa_q}{seq_{cpu} + pa_{cpu} + seq_{io} + pa_{io}} \quad (9)$$

Based on Equation 7, the ideal speedup model is depicted in Figure 1. The model shows a rather optimistic speedup behavior, omitting any inter-processor or inter-node communication overhead, thread synchronization issues, and I/O buffering scenarios.

Figure 1: Ideal (Theoretical) Speedup



Note: The X-axis depicts the number of CPU's, the Y-axis denotes the number of disks, and the Z-axis represents the speedup. To calibrate the model, the parameters $\psi(cpu)$ and $\psi(io)$ were set to 0.05, whereas the parameters $\omega(cpu)$ and $\omega(io)$ were set to 0.4, respectively.

4 Application Communication Patterns

In this section, the study introduces three capacity functions that are based on different application communication behaviors. For each of the different capacity functions, the performance behavior is presented and quantified via an empirical analysis on a symmetric multi processing (SMP) system. Hence, this section establishes the baseline for quantifying the speedup behavior for parallel applications on cluster based systems. The study discusses three capacity models, that based on particular application communication patterns provide realistic expectations with regard to SMP scaling. Research conducted by Artis [13], who introduced the Geometric capacity model, Kraus [14] and Hennessy et al. [4] (detailed discussions on Amdahl's law), and Gunther [22] (comprehensive definition and analysis of the Geometric capacity model) established the actual research baseline.

As previously mentioned, Amdahl's Law states that there is always some irreducible part of an application that can not be decomposed any further to run in parallel. The implication is that the greater the parallelism the greater the impact of the serial portion on the actual scaleup. Kraus, as well as Hennessy & Patterson, discuss in great details the implications that Amdahl's law has on SMP systems [4][14]. Based on these comprehensive studies, the Amdahl capacity model can be formulated as:

$$A(p) = \frac{p}{1 + \alpha(p-1)} \quad (10)$$

Equation 10 incorporates a serialization factor α $[0 < \alpha < 1]$, and defines the number of available processors on the SMP system as p. The serialization factor refers to the serial fraction of the workload that has to be processed sequentially. As $p \rightarrow \infty$, the SMP capacity approaches the asymptote $A(\infty) \cong 1/\alpha$. As an example, with a serialization factor of only 2%, the effective capacity of a 12-way SMP system can be predicted to be approximately 9.84. The ramification is that 2.16 CPU equivalents in capacity can not be utilized. The hidden dynamics of Amdahl's capacity law reveal a model that is based on a broadcast protocol. See Gunther for a mathematical analysis of the Amdahl capacity function [22]. The suspension of any computation activities during the broadcast cycle causes a serialization effect on the processors, which impacts the aggregate response time. The number of times the *message to be transferred* has to be setup is not a function of the communication protocol, but is a constant that is set to 1.

The Geometric capacity model [13], [22] is outlined in Equation 11 and Equation 11a, and can be defined as the summation of a geometric series:

$$E(p) = \sum_{n=0}^{p-1} \phi^n \quad (11) \qquad E(p) = \frac{1 - e^{p \cdot \ln(\phi)}}{1 - \phi} \quad (11a)$$

In this capacity model, the coefficient ϕ is defined as the MP factor, reflecting the fraction of single processor capacity *available* as each CPU (p) is added to the SMP system [$0 < \phi < 1$]. With an MP factor of 98% ($\phi = 0.98$), a 12-way SMP system has a predicted capacity of approximately 10.75, which implies that 1.25 CPU equivalents are being consumed by (SMP) computational overhead. Interestingly enough, analyzing the dynamics of the Geometric scaling function reveals that its communication behavior corresponds to a nearest neighbor protocol [16]. Therefore, a linear chain of ($p - 1$) bi-directional communication links can be used to represent the overhead in a p-way parallel system. The implication is that as the number of CPU's is increased in the configuration, the overhead per single message grows as well. This is due to the fact that every CPU (from the source to the destination processor) has to pass the message along the way. Counterbalancing the increased cost per message is the fact that fewer message links (compared to the Amdahl capacity models) have to be established while scaling the number of CPU's. The argument is that the Geometric scaling function is best suited for an hyper-cube architecture, where all the communication activities occur via hops between nearest neighbors in a respective cube [4] [8].

While studying scientific and mathematical parallel applications [6][8][15] [17], an in-depth analysis revealed that in many circumstances a *request-response* based application communication behavior prevailed, where a single application thread only communicated with a subset of the other available threads. In the majority of the experiments, the number of communication threads stabilized around the number of available CPU communication links. Therefore, a communication-link based capacity model was introduced that represents a communication overhead behavior as formulated in Equation 12. To reiterate, the capacity function outlined in Equation 12 is suited for parallel applications that disclose a communication behavior where an application thread only communicates with a subset of the other application threads that are executed simultaneously. The communication-link model reveals some similarity to the quadratic scaleup model introduced by Gunther [22] for commercial database systems, as it also specifies a maximum capacity limit. It differentiates itself though through a thread subset based request-response communication protocol. The model's capacity overhead factor h refers to the capacity reduction that is introduced by every additional CPU in the system [$0 < h < 1$]. To illustrate, by setting the *h factor* to 2%, the predicted capacity of a 12-way SMP system equals to approximately 10.68, outlining that 1.32 CPU equivalents are not being utilized to process the workload.

$$H(p) = p - h \cdot \frac{p(p-1)}{2} \quad (12)$$

To further elaborate on the different communication behaviors simulated via the capacity functions, the study conducted an empirical analysis of the capacity behavior of the Barnes-Hut application and the LU Kernel, respectively [4]. The benchmark environment consisted of an SMP system that was configured with 1GB of memory, supporting a hardware setup with up to 16 (180 MHz) processors. The measurements were taken on five different hardware configurations, starting with one processor and activating additional CPU's in each test run, until the system was configured with 16 processors. The SMP capacity values in Figure 2 and 3 are reported relative to the single processor version of both benchmark programs. Both benchmark programs were tuned to improve the memory behavior.

The Barnes-Hut application is based on an algorithm presented in [18]. The algorithm is widely being used in the field of astrophysics. The application simulates the evolution of a system of bodies under the influence of gravitational forces. It is a classical gravitational n-body simulation, in which each body is modeled as a point mass. The simulation proceeds over time-steps, each step computing the net force on every body, and thereby updating that body's position (as well as some other attributes). The data structure used is based on a hierarchical octree representation of space in three dimensions (each node has up to eight children). Obtaining effective parallel performance in this scenario is rather challenging, as the distribution of the bodies is non-uniform (and changes over time). This implies that partitioning the work among the processors (and maintaining a high level of locality of reference) is a rather daunting task. As gravitational forces fall off rather quickly, each cell requires touching a smaller subset of other cells, a behavior that drives the communication overhead. To reiterate, the Barnes-Hut application depicts a parallel construct where over time, the interaction among the bodies falls off. From an implementation perspective, allocating a subtree to each processor in the configuration allows partitioning the octree.

Figure 2: Barnes-Hut Application

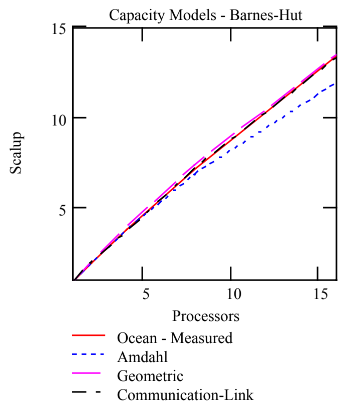


Figure 2 illustrates that the Communication-link model captures the SMP capacity behavior with a description error of 1.4% and 0.7% for the eight and 16 CPU configurations, respectively. While the Geometric model is basically in line with the Communication-link capacity function, the Amdahl model misstates the SMP capacity by a substantial margin. Within the eight CPU configuration, the Amdahl capacity model prediction error was 4.7%. For the 16 CPU setup, the model prediction error was 11.2%. The ramification is that for the Barnes-Hut application on the given hardware, the Communication-link capacity model captures the capacity behavior, and therefore the thread communication pattern in the most effective way.

The LU Kernel represents a LU factorization of a dense matrix [4] [17]. For a ($n * n$) matrix, the running time equals to n^3, whereas the parallelism is proportional to n^2. Dense LU factorization can efficiently perform by decomposing (*blocking*) the algorithm, which leads to a highly efficient cache behavior and a rather low communication overhead. The block size is chosen (1) to be large enough to insure a low cache miss rate and (2) small enough to reduce the time spent in the less parallel parts of the computation. The dense matrix multiplication is performed on the processor that owns the destination block. The ramification is that the communication behavior is regular and predictable. In other words, the communication is proportional to the boundary of a block, and therefore scales with the data set size at a rate proportional to the side of a square with n points. Therefore, the Amdahl capacity function is suited to conduct a sensitivity study.

Figure 3: LU Kernel

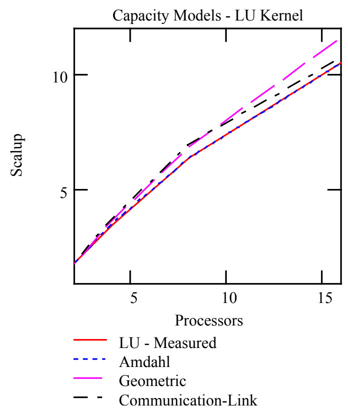
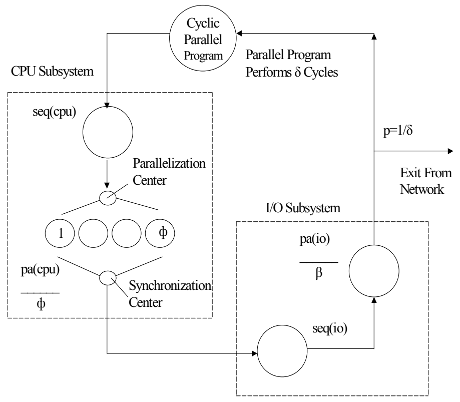


Figure 3 reveals that while the Amdahl capacity function accurately predicts the capacity behavior, the Communication-link as well as the Geometric model overstate the capacity behavior of the LU kernel. Across the entire CPU setup (from 1 to 16 CPU's), the mean prediction error for the Amdahl model was 0.9%, compared to a prediction error of 5% and 8.2% for the Communication-link and the Geometric models, respectively.

5 Thread Synchronization

Based on the assumption that the execution time of the processing nodes are independent of each other, this study argues that it is imperative to incorporate into the model the fact that the different CPU's finish processing their requests at different points in time. Many parallel scientific applications that can be partitioned into a parallel CPU and a parallel I/O phase synchronize the worker threads after the CPU phase (see Figure 4). The ramification is that the total execution time of the CPU phase is dominated by the slowest processor execution time. Based on work in [5][20][21], the study suggests to model such a behavior by substituting the basic parallel CPU component (that is expressed as *pa(cpu) / ϕ*) with a delay center *dlyctr(ϕ)*. In queuing theory, a delay center describes the scenario where a request is serviced immediately [5].

Figure 4: Queuing Model with Synchronization Overhead



As a task finishes the sequential phase of the CPU cycle, it reaches an actual *parallelization center* where the task is decomposed into *ϕ* jobs that proceed through the delay center.

$$\text{cpusync}_1(\phi,\beta) = \delta \cdot \left(\text{seq}_{cpu} + \frac{\text{dlyctr}(\phi)\ \text{pa}_{cpu}}{\phi} + \text{seq}_{io} + \frac{\text{pa}_{io}}{\beta} \right) \quad (13)$$

Where the delay factor is defined as:

$$\text{dlyctr}(\phi) = \sum_{i=1}^{\phi} \frac{1}{i^2} \quad (14)$$

Resulting into a speedup definition of:

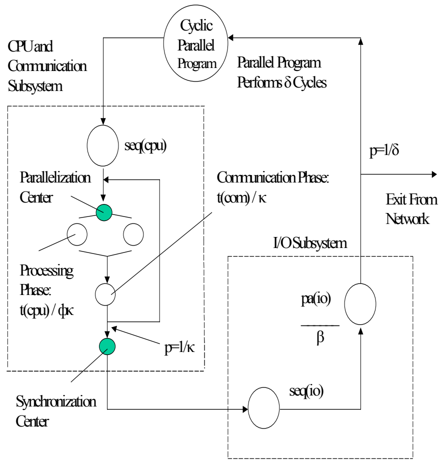
$$\text{spdp}_1(\phi,\beta) = \frac{1}{\psi_{cpu} + \frac{\text{dlyctr}(\phi)\cdot\omega_{cpu}}{\phi} + \psi_{io} + \frac{\omega_{io}}{\beta}} \quad (15)$$

The synchronization center collects the processed tasks, and at the point in time where the number of received tasks equals to *ϕ*, the system substitutes the parallel sub-tasks with a new job that represents the global execution of the parallel program. In this study, the distribution of the delays is exponentially distributed (with a mean distribution of *pa(cpu) / ϕ*). The execution time of a parallel application (with CPU synchronization) and the speedup behavior are expressed in Equation 10 and Equation 11, respectively. It is emphasized that the introduction of a thread synchronization mechanism results in a more conservative speedup prediction behavior compared to the ideal speedup curve expressed in Figure 1.

5.1 Internode Communication

In a next phase, this study incorporates the different inter-node communication behaviors as discussed in Section 4 into the speedup equation. The rationale is that for a variety of scientific parallel applications, the parallel portion of the CPU phase *pa(cpu)* can be decomposed into *κ* distinct intervals. Each interval can further be decomposed into (1) a computation phase that is followed by (2) a communication phase [6]. This study defines the total computation time as *t(cpu)*, and the total communication time as *t(com)*. The interconnect among the nodes can be modeled as a queuing center, with a service time distribution that represents the time the worker threads are communicating among each other.

Figure 5: Queuing Network with Communication Contention



While scaling the number of CPU's (the first order model assumes that *α = ϕ*), the thesis made is that the execution time of the computational phase scales with the number of available CPU's. In other words, the average per

CPU computation time can be expressed as *(t(cpu) / ϕ)*. On the other hand, the scaling factor for the communication phase depends (1) on the distributed data structures that are being incorporated into the scientific parallel application [6], and (2) on the resulting inter-node communication pattern. The study proposes a first-order scaling factor *comsf(ϕ),* which allows formulating the average communication time per CPU phase as (*t(com)* * *comsf(ϕ)*). As depicted in Figure 5, each task steps first through an infinite server with an average delay of *t(cpu) / ϕ*κ*. In a next phase, the task proceeds to the queuing center that represents the communication phase. Each task computes on average κ cycles, leaves the closed queuing network, and arrives at the synchronization center. At the point where all the tasks have arrived at the synchronization center, a single thread (representing the global program computation phase) substitutes all the sub-tasks.

This study utilizes a Mean Value Analysis (MVA) based approach to formulate and quantify the time spent in the CPU phase (any time spent between the parallization and the synchronization centers, respectively). The proposed MVA based approach is closely related to research studies conducted by Jan [5], Stone [7], Koenigs [6], Gennaro [20], Cremonesi [21], and Zahorjan [12], respectively. The inter-node communication patterns analyzed represent (1) the broadcast, (2) the Communication-link, and (3) the next neighbor protocols, respectively. The generic speedup equation that is utilized to quantify the different communication behaviors is depicted in Equation 16. The 3 distinct communication behaviors were captured (and therefore simulated) via accordingly adjusting the *comsf(ϕ)* factor.

$$\text{cluster\_lnk}_{\phi,\beta} = \frac{1}{\text{cpu} + \text{com} + \text{io}} \qquad (16)$$

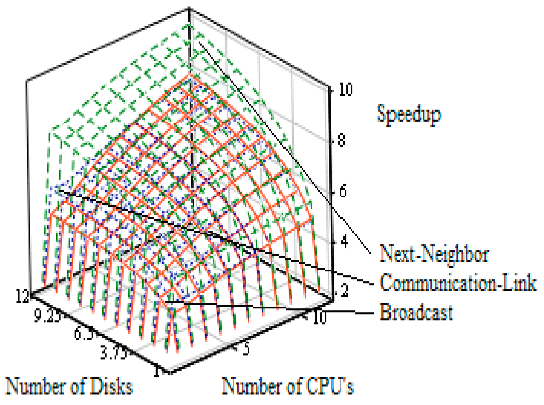
$$\text{cpu} = \text{seq\_cpu} + \text{dlyctr}(\phi) \cdot \frac{\text{pa\_cpu}}{\phi} \qquad (17)$$

$$\text{com} = \text{comsf}(\phi) \cdot \frac{\text{pa\_com}}{\phi} \qquad (18)$$

$$\text{io} = \text{seq\_io} + \frac{\text{pa\_io}}{\beta} \qquad (19)$$

Based on the set of calibration parameters used in this experiment (the actual workload definition), the conducted simulations revealed that the next-neighbor based communication behavior was able to outperform (from a relative scalability perspective) the other two communication scenarios (see Figure 6). Further, based on the simulated scenario, the Communication-link model slightly edged the broadcast protocol. This experiment outlines the profound impact that the application communication behavior has on the speedup of cluster based systems. Therefore, the study argues that in order to obtain additional insights into the actual performance characteristics of cluster systems, it is imperative to go beyond the normally used (and rather trivial) latency and bandwidth determination efforts. More specifically, it is important to consider the performance impact that the application has with regard to features such as remote memory access operations, completion notification, and address translation mechanisms.

Figure 6: Speedup Behavior – Cluster



Note: The X-axis depicts the number of CPU's, the Y-axis denotes the number of disks, and the Z-axis represents the speedup. To calibrate the model, the parameters *ψ(cpu)* and *ψ(io)* were set to 0.02 and 0.05, whereas the parameters *ω(cpu)* and *ω(io)* were set to 0.4 and 0.38, respectively. The parameter *pa_com* was set to 0.15

5 Conclusions and Future Work

Any system design effort includes a qualitative performance analysis to determine the behavior and efficiency of the system environment under consideration. Normally, assumptions have to be formulated and agreed upon to establish an experimental baseline. The actual analysis is conducted by either utilizing an analytical model, a simulation model, or an empirical analysis based approach. The three different techniques essentially represent different *stages* in the performance analysis spectrum, describing different levels of modeling accuracy, cost of constructing the model or benchmark, and forecasting capability of the chosen methodology.

This study showed that even with a fully controlled synthetic workload, modeling the speedup behavior of a scientific application in a cluster environment is a daunting task. The approach taken is based on queuing networks and utilizes a Mean Value Analysis to extract the qualitative as well as quantitative behavior of a scientific application. The simulated execution pattern analyzed in this study can be described as consisting of a sequence of cyclic CPU and communication requests followed by an actual I/O phase. The focus of the introduced model was to illustrate the (relative) impact that CPU and I/O parallelism has on the speedup behavior based on the three different inter-node communication scenarios: broadcast, nearest-neighbor, and request-response. As a future work item, it would be interesting to expand on the proposed MVA models and utilize a convolution algorithm to conduct a probability analysis. An MVA based analysis provides algorithms that allow determining the average queue length or average response time of various components, but does not allow studying the distribution or variance of the queue length or the response time per se.

References

1. *Amdahl, G., "On the Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", Proceeding of the AFIPS Conference, 1967*

2. *DeWitt, D., Gray, J., "Parallel Database Systems: The Future of High Performance Database Systems", 1992*

3. *Eager, D., Zahorjan, J., Lazowska E., "Speedup verses Efficiency in Parallel Systems", IEEE, 1989*

4. *Hennessy, J., Patterson, D., "Computer Architecture, a Quantitative Approach", Third Edition, Morgan Kaufmann, 2002.*

5. *Jain, R., "The Art of Computer System Performance Analysis", John Wiley, 1991*

6. *Koenigs, A. "Industrial Strength Parallel Computing", Morgan Kaufmann, San Francisco, 2000*

7. *Stone, H., "High Performance Computers", Addison-Wesley, 1st Edition, New York, NY, 1991*

8. *Pfister, G. F., "In Search of Clusters", Second Edition, Prentice Hall PTR, NY, 1998.*

9. *Plyzos, G., Pasquale, B., "A Static Analysis of I/O Characterization of Scientific Applications in a Production Workload", Proceedings of Supercomputing 1993.*

10. *Schmuck, F., Haskin, R., "GPFS – A Shared Disk File System for Large Computing Clusters", USENIX FAST 02, Monterey, CA, 2002.*

11. *Wu, C., Baylor, S., "Parallel I/O Workload Characteristics using Vesta", I/O in Parallel and Distributed Computer Systems, Kluwer Academic Publisher, 1996.*

12. *Zahorjan, J., Lazowska, E., Graham, G., Sevcik, K., "Quantitative System Performance – Computer System Analysis Using Queuing Network Models", Prentice Hall, NJ, 1984*

13. *Artis, H.P., "Quantifying Multiprocessor Overhead", CMG91 Conference, 1991*

14. *Kraus, I.F., "The Role of Symmetry in the Parallel Sysplex", Proceedings CMG95 Conference, 1995*

15. *Scali "NAS Parallel Benchmarks on a Scali System" http://www.scali.com, 2000*

16. *Volmer S., "Fast Approximate Nearest Neighbor Queries in Metric Feature Space by Buoy Indexing" FICG, 2001*

17. *Woo, S., Ohara, M., Torrie, E., Singh, J., Gupta, A., "SPLASH-2 Programs, Characterization & Methodological Considerations", 22nd ISCA, 1995*

18. *Barnes, J., Hut, P., "A Hierarchical O(n log n) force calculation algorithm", 1986*

19. *Braam, P., Zahir, R., "Luster Technical Project Summary", Cluster File Systems, 2001*

20. *Gennaro, C., "Performance Model for I/O Bound SPMD Applications on Clusters of Workstations", Politecnico di Milano, 1999*

21. *Cremonesi, P., Gennaro, C., "Integrated Performance Models for SPMD Applications and MMD Architectures", IEEE Trans. Parallel & Distributed Systems, Vol.13, 2002*

22. *Gunther, N., "The Practical Performance Analyst", McGraw Hill, 2000*