INTERNATIONAL JOURNAL OF CONTEMPORARY ACADEMICS (IJCA) ISSN 2536-7110 © 2022 VOL. 6 (1), Pp.80-88, © MAY, 2022 HIGH THROUGHPUT COMPUTING: MECHANISMS AND CHALLENGES

BY

Ogundoju B. M., Idowu C. S., Asunbiaro S. C. and Abiodun O. A.

www.guildaa.org

HIGH THROUGHPUT COMPUTING: MECHANISMS AND CHALLENGES

BY

Ogundoju B. M., Idowu C. S., Asunbiaro S. C. and Abiodun O. A.

Department of Computer Science,

Adeyemi College of Education, Ondo

Abstract

In communication networks, such as Ethernet or packet radio, throughput or network throughput is the average rate of successful message delivered over a communication channel. This data may be delivered over a physical or logical link, or pass through a certain network node. The goal of most computing operations is better or improved throughput and concerns are mostly about how many floating point operations can be extracted from the computing environment per week rather than the number of such operations the environment can provide per week. High Throughput Computing (HTC) is used to describe a situation where a large amount of computational power is delivered over a long period of time. This paper focuses on mechanisms, challenges and security of High Throughput Computing. The paper also drew distinction between High Throughput Computing (HTC), High Performance Computing (HPC) and Many Task Computing (MTC).

Keywords: Throughout, Computing, Challenges, Mechanism

Introduction

For many research and engineering projects, the quality of the research or the product is heavily dependent upon the quantity of computing cycles available. It is not uncommon to find problems that require weeks or months of computation to solve. Gustavson, D.B. and Li, Q. (1995) asserted that scientists and engineers engaged in this sort of work need a computing environment that delivers large amounts of computational power over a long period of time. Such an environment is called a High Throughput

BY

Ogundoju B. M., Idowu C. S., Asunbiaro S. C. and Abiodun O. A.

www.guildaa.org

Computing (HTC) environment. In contrast, High Performance Computing (HPC) environments deliver a tremendous amount of computing power over a short period of time. HPC environments are often measured in terms of FLoating point Operations Per Second (FLOPS). A growing community is not concerned about operations per second, but operations per month or per year. Their problems are of a much larger scale. They are more interested in how many jobs they can complete over a long period of time instead of how fast an individual job can complete.

The key to HTC is to efficiently harness the use of all available resources. Years ago, the engineering and scientific community relied on a large, centralized mainframe or a supercomputer to do computational work. A large number of individuals and groups needed to pool their financial resources to afford such a machine. Users had to wait for their turn on the mainframe, and they had a limited amount of time alloted. While this environment was inconvenient for users, the utilization of the mainframe was high; it was busy nearly all the time. High-throughput computing (HTC) is a Computer Science term to describe the use of many computing resources over long periods of time to accomplish a computational task. Kerbyson et al., (2004).

As computers became smaller, faster, and cheaper, users moved away from centralized mainframes and purchased personal desktop workstations and PCs. An individual or small group could afford a computing resource that was available whenever they wanted it. The personal computer is slower than the large centralized machine, but it provides exclusive access. Now, instead of one giant computer for a large institution, there may be hundreds or thousands of personal computers. This is an environment of distributed ownership, where individuals throughout an organization own their own resources. The total computational power of the institution as a whole may rise dramatically as the result of such a change, but because of distributed ownership, individuals have not been able to capitalize on the institutional growth of computing power. And, while distributed ownership is more convenient for the users, the utilization of the computing power is lower. Many personal desktop machines sit idle for very long periods of time while their owners are busy doing other things (such as being away at lunch, in meetings, or at home sleeping).

The System Throughput

The **system throughput** or **aggregate throughput** is the sum of the data rates that are delivered to all terminals in a network. The throughput can be analyzed mathematically by means of queuing theory, where the load in packets per time unit is denoted arrival rate λ, and the throughput in packets per time unit is denoted departure rate μ.

Maximum Throughput

Users of telecommunications devices, systems designers, and researchers into communication theory are often interested in knowing the expected performance of a system. From a user's perspective, this is often phrased as either "which device will get my data there most effectively for my needs?", or "which device will deliver the most data per unit cost?". Systems designers are often interested in selecting the most effective architecture or design constraints for a system, which drive its final performance. In most cases, the benchmark of what a system is capable of, or its 'maximum performance' is what the user or designer is interested in. Four different values have meaning in the context of "maximum throughput", used in comparing the 'upper limit' conceptual performance of multiple systems. They are

'maximum theoretical throughput', 'maximum achievable throughput', and 'peak measured throughput' and 'maximum sustained throughput'. These represent different quantities and care must be taken that the same definitions are used when comparing different 'maximum throughput' values. Comparing throughput values is also dependent on each bit carrying the same amount of information. Data compression can significantly skew throughput calculations, including generating values greater than 100%. If the communication is mediated by several links in series with different bit rates, the maximum throughput of the overall link is lower than or equal to the lowest bit rate. The lowest value link in the series is referred to as the bottleneck.

Maximum Theoretical Throughput

This number is closely related to the channel capacity of the system, and is the maximum possible quantity of data that can be transmitted under ideal circumstances. In some cases this number is reported as equal to the channel capacity, though this can be deceptive, as only non-packetized systems (asynchronous) technologies can achieve this without data compression. Maximum theoretical throughput is more accurately reported to take into account format and specification overhead with best case assumptions. This number, like the closely related term 'maximum achievable throughput' below, is primarily used as a rough calculated value, such as for determining bounds on possible performance early in a system design phase.

Peak Measured Throughput

The above values are theoretical or calculated values. Peak measured throughput is throughput measured by a real, implemented system, or a simulated system. The value is the throughput measured over a short period of time; mathematically, this is the limit taken with respect to throughput as time approaches zero. This term is synonymous with "instantaneous throughput". This number is useful for systems that rely on burst data transmission, however, for systems with a high duty cycle this is less likely to be a useful measure of system performance.

Maximum Sustained Throughput

This value is the throughput averaged or integrated over a long time (sometimes considered infinity). For high duty cycle networks this is likely to be the most accurate indicator of system performance. The maximum throughput is defined as the asymptotic throughput when the load (the amount of incoming data) is very large. In packet switched systems where the load and the throughput always are equal (where packet loss does not occur), the maximum throughput may be defined as the minimum load in bit/s that causes the delivery time (the latency) to become unstable and increase towards infinity. This value can also be used deceptively in relation to peak measured throughput to conceal packet shaping.

Channel Utilization - Channel Efficiency - Normalized Throughput

Throughput is sometimes normalized and measured in percentage, but normalization may cause confusion regarding what the percentage is related to. Channel utilization, Channel efficiency and packet drop rate in percentage are less ambiguous terms.

The channel efficiency, also known as bandwidth utilization efficiency, in percentage is the achieved throughput related to the net bitrate in bit/s of a digital communication channel. For example, if the throughput is 70 Mbit/s in a 100 Mbit/s

Ethernet connection, the channel efficiency is 70%. In this example, effective 70Mbits of data are transmitted every second.

Channel utilization is instead a term related to the use of the channel disregarding the throughput. It counts not only with the data bits but also with the overhead that makes use of the channel. The transmission overhead consists of preamble sequences, frame headers and acknowledge packets. The definitions assume a noiseless channel. Otherwise, the throughput would not be only associated to the nature (efficiency) of the protocol but also to retransmissions resultant from quality of the channel. In a simplistic approach, channel efficiency can be equal to channel utilization assuming that acknowledge packets are zero-length and that the communications provider will not see any bandwidth relative to retransmissions or headers. Therefore, certain texts mark a difference between channel utilization and protocol efficiency.

Mechanism for High Throughput Computing

There are many mechanisms for high throughput computing, the Condor's power is one of the popular mechanisms and shall be fully discussed. Other mechanisms include ClassAds, Remote System Calls and Check-pointing mechanism.

The Condor's Power

Condor is a software system that creates a High Throughput Computing (HTC) environment. It effectively utilizes the computing power of workstations that communicates over a network. Condor can manage a dedicated cluster of workstations. Its power comes from the ability to effectively harness non-dedicated, pre-existing resources under distributed ownership.

A user submits the job to Condor. Condor finds an available machine on the network and begins running the job on that machine. Condor has the capability to detect that a machine running a Condor job is no longer available (perhaps because the owner of the machine came back from lunch and started typing on the keyboard). It can checkpoint the job and move (migrate) the jobs to a different machine which would otherwise be idle. Condor continues job on the new machine from precisely where it left off.

In those cases where Condor can checkpoint and migrate a job, Condor makes it easy to maximize the number of machines which can run a job. In this case, there is no requirement for machines to share file systems (for example, with NFS or AFS), so that machines across an entire enterprise can run a job, including machines in different administrative domains.

Condor can be a real time saver when a job must be run many (hundreds of) different times, perhaps with hundreds of different data sets. With one command, all of the hundreds of jobs are submitted to Condor. Depending upon the number of machines in the Condor pool, dozens or even hundreds of otherwise idle machines can be running the job at any given moment.

Condor does not require an account (login) on machines where it runs a job. Condor can do this because of its *remote system call* technology, which traps library calls for such operations as reading or writing from disk files. The calls are transmitted over the network to be performed on the machine where the job was submitted.

Condor provides powerful resource management by match-making resource owners with resource consumers. This is the cornerstone of a successful HTC environment. Other computer cluster resource management systems attach

properties to the job queues themselves, resulting in user confusion over which queue to use as well as administrative hassle in constantly adding and editing queue properties to satisfy user demands. Condor implements *ClassAds*, a clean design that simplifies the user's submission of jobs.

ClassAds work in a fashion similar to the newspaper classified advertising want-ads. All machines in the Condor pool advertise their resource properties, both static and dynamic, such as available RAM memory, CPU type, CPU speed, virtual memory size, physical location, and current load average, in a *resource offer* ad. A user specifies a *resource request* ad when submitting a job. The request defines both the required and a desired set of properties of the resource to run the job. Condor acts as a broker by matching and ranking resource offer ads with resource request ads, making certain that all requirements in both ads are satisfied. During this match-making process, Condor also considers several layers of priority values: the priority the user assigned to the resource request ad, the priority of the user which submitted the ad, and desire of machines in the pool to accept certain types of ads over others.

The Condor's Features

Checkpoint and Migration.

Where programs can be linked with Condor libraries, users of Condor may be assured that their jobs will eventually complete, even in the ever changing environment that Condor utilizes. As a machine running a job submitted to Condor becomes unavailable, the job can be check pointed. The job may continue after migrating to another machine. Condor's periodic checkpoint feature periodically checkpoints a job even in lieu of migration in order to safeguard the accumulated computation time on a job from being lost in the event of a system failure such as the machine being shutdown or a crash.

Remote System Calls.

Despite running jobs on remote machines, the Condor standard universe execution mode preserves the local execution environment via remote system calls. Users do not have to worry about making data files available to remote workstations or even obtaining a login account on remote workstations before Condor executes their programs there. The program behaves under Condor as if it were running as the user that submitted the job on the workstation where it was originally submitted, no matter on which machine it really ends up executing on.

No Changes Necessary to User's Source Code.

No special programming is required to use Condor. Condor is able to run non-interactive programs. The checkpoint and migration of programs by Condor is transparent and automatic, as is the use of remote system calls. If these facilities are desired, the user only re-links the program. The code is neither recompiled nor changed.

Pools of Machines can be Hooked Together.

Flocking is a feature of Condor that allows jobs submitted within a first pool of Condor machines to execute on a second pool. The mechanism is flexible, following requests from the job submission, while allowing the second pool, or a subset of machines within the second pool to set policies over the conditions under which jobs are executed.

Jobs can be Ordered.

The ordering of job execution required by dependencies among jobs in a set is easily handled. The set of jobs is specified using a directed acyclic graph, where each job is a node in the graph. Jobs are submitted to Condor following the dependencies given by the graph.

Condor Enables Grid Computing.

As grid computing becomes a reality, Condor is already there. The technique of glide-in allows jobs submitted to Condor to be executed on grid machines in various locations worldwide. As the details of grid computing evolve, so does Condor's ability, starting with Glob-us-controlled resources.

Sensitive to the Desires of Machine Owners.

The owner of a machine has complete priority over the use of the machine. An owner is generally happy to let others compute on the machine while it is idle, but wants it back promptly upon returning. The owner does not want to take special action to regain control. Condor handles this automatically.

ClassAds.

The ClassAd mechanism in Condor provides an extremely flexible, expressive framework for matchmaking resource requests with resource offers. Users can easily request both job requirements and job desires. For example, a user can require that a job run on a machine with 64 Mbytes of RAM, but state a preference for 128 Mbytes, if available. A workstation owner can state a preference that the workstation runs jobs from a specified set of users. The owner can also require that there be no interactive workstation activity detectable at certain hours before Condor could start a job. Job requirements/preferences and resource availability constraints can be described in terms of powerful expressions, resulting in Condor's adaptation to nearly any desired policy.

Current Limitations

Limitations on Jobs which can be Check pointed

Although Condor can schedule and run any type of process, Condor does have some limitations on jobs that it can transparently checkpoint and migrate:

1. Multi-process jobs are not allowed. This includes system calls such as fork(), exec(), and system().
2. Inter-process communication is not allowed. This includes pipes, semaphores, and shared memory.
3. Network communication must be brief. A job *may* make network connections using system calls such as socket(), but a network connection left open for long periods will delay check-pointing and migration.
4. Sending or receiving the SIGUSR2 or SIGTSTP signals is not allowed. Condor reserves these signals for its own use. Sending or receiving all other signals *is* allowed.
5. Alarms, timers, and sleeping are not allowed. This includes system calls such as alarm(), get-timer(), and sleep().
6. Multiple kernel-level threads are not allowed. However, multiple user-level threads *are* allowed.

7. Memory mapped files are not allowed. This includes system calls such as mmap() and munmap().
8. File locks are allowed, but not retained between checkpoints.

Note: these limitations *only* apply to jobs which Condor has been asked to transparently checkpoint. If job check-pointing is not desired, the limitations above do not apply.

Security Implications.

Condor does a significant amount of work to prevent security hazards, but loopholes are known to exist. Condor can be instructed to run user programs only as the UNIX user nobody, a user login which traditionally has very restricted access. But even with access solely as user nobody, a sufficiently malicious individual could do such things as fill up /tmp (which is world writable) and/or gain read access to world readable files. Furthermore, where the security of machines in the pool is a high concern, only machines where the UNIX user root on that machine can be trusted should be admitted into the pool. Condor provides the administrator with extensive security mechanisms to enforce desired policies.

Jobs Need to be Re-linked to get Check-pointing and Remote System Calls

Although typically no source code changes are required, Condor requires that the jobs be re-linked with the Condor libraries to take advantage of check-pointing and remote system calls. This often precludes commercial software binaries from taking advantage of these services because commercial packages rarely make their object code available. Condor's other services are still available for these commercial packages.

Challenges

The HTC community is also concerned with robustness and reliability of jobs over a long-time scale. That is, being able to create a reliable system from unreliable components. Some HTC systems, such as Condor can run tasks on opportunistic resources. It is a difficult problem, however, to operate in this environment. On one hand the system needs to provide a reliable operating environment for the user's jobs, but at the same time the system must not compromise the integrity of the execute node and allow the owner to always have full control of their resources.

High-throughput computing vs. high-performance computing vs. many-task computing

There are many differences between **high-throughput computing**, high-performance computing (HPC), and many-task computing (MTC). HPC tasks are characterized as needing large amounts of computing power for short periods of time, whereas HTC tasks also require large amounts of computing, but for much longer times (months and years, rather than hours and days). HPC environments are often measured in terms of FLOPS. The HTC community, however, is not concerned about operations per second, but rather operations per month or per year. Therefore, the HTC field is more interested in how many jobs can be completed over a long period of time instead of how fast an individual job can complete. Geist et al (1994).
As a general rule, HPC systems are tightly coupled parallel jobs, and as such they must execute within a particular site with low-latency interconnects. Conversely, HTC systems are independent, sequential jobs that can be individually scheduled on many different computing resources across

multiple administrative boundaries. HTC systems achieve this using various grid computing technologies and techniques.

MTC aims to bridge the gap between HTC and HPC. MTC is reminiscent of HTC, but it differs in the emphasis of using many computing resources over short periods of time to accomplish many computational tasks (i.e. including both dependent and independent tasks), where the primary metrics are measured in seconds (e.g. FLOPS, tasks/s, MB/s I/O rates), as opposed to operations (e.g. jobs) per month. MTC denotes high-performance computations comprising multiple distinct activities, coupled via file system operations.

Conclusions

In this paper we briefly described the mechanism, challenges and security of High Throughput Computing. More emphasis was laid on Condor (a software system that creates a High Throughput Computing). Condor can provide resource management services which efficiently harness the use of all available resources. A brief distinction was drawn between High Throughput Computing (HTC), High Performance Computing (HPC) and Many Task Computing (MTC).

References

Amza C., Cox, A. L., Dwarkadas S,. Keleher,P., Rajamony R. and W. Zwaenepoel. (2006)

Thread Marks: Shared memory computing on networks of workstations.

Blahut R. E. (2004) *Algebraic Codes for Data Transmission* Cambridge University Press,

ISBN 0-521-55374-1

Boden, N. J., Cohen, D., Felderman, R.E., Kulawik, A. E., Seitz C. L., Seizovic, J. N.and Wen-

King Su,(1995). *Myrinet -- A Gigabit-per-second Local Area Network*, IEEE Micro **15**,

No. 1, Jan., 29--36.

C97 condor team, "The condor High Throughput Computing " *http;//www.cs.wisc.edu/condor/*

Dunigan, T.H.,. Fahey, M.R., White, J.B.and Worley, P.H (2003). *Early Evaluation of the Cray*

X1, Proc. SC2003, Phoenix, AZ, USA.

Flynn, M.J. (1972). *Some computer organizations and their effectiveness*, IEEE Transactions on

Computing, **C-21**, 948-960.

Geist, A., Beguelin, A. Dongarra, J. Manchek, V Jibing, W. and. Sunderam, V(1994). *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Boston,

Gustavson, D.B.and Li, Q. (1995) *Local-Area MultiProcessor: the Scalable Coherent Interface.*, SCIzzL Report, Santa Clara University, Dept. of Computer Engineering.

Available through: www.scizzle.com.

L87 M. Litzko, "Remote Unix-Turning Idle Workstations into Cycle Services*," in proceedings*

of Usnix summer Conference, 1987.

Johnson, Graham, (1973) *High Speed Digital Design, a Handbook of Black Magic*, Prentice Hall,

1973, ISBN 0-13-395724-1

Kerbyson, D.J., Hoisie A, Pakin, S. Petrini, F., Wassermann, H.J. (2004). *A Performance*

Evaluation on an Alpha EV7 Processing Node, Int. J. of High Perf. Comp. Appl., **18**(2), 199--209, Summer .

Li H. H. (2005) "Impact of Lossy Links on Performance of Multihop Wireless

Networks", IEEE, Proceedings of the 14th International Conference on Computer

Communications and Networks, Oct 2005, 303 - 308

Nagel, W. E.(1998)..*Applications on the Cenju: First Experience with Effective Performance*,

2nd Cenju Workshop, Sankt Augustin, Germany. October 1998.

Rappaport T. S. (2002). *Wireless Communications, Principles and Practice* second

edition, Prentice Hall, , ISBN 0-13-042232-0

Roddy D. (2001) *Satellite Communications* third edition, McGraw-Hill,

ISBN 0- 07-137176-1, 202 - 207