Parallel Processing over a Peer-to-Peer Network

Constructing the Poor Man's Supercomputer

Karl Fenech
Department of Computer Science
Faculty of ICT
University of Malta
karl.fenech@gmail.com

Kevin Vella
Department of Computer Science
Faculty of ICT
University of Malta
kevin.vella@um.edu.mt

Abstract

The aggregation of typical home computers through a peer-to-peer (P2P) framework over the Internet would yield a virtual supercomputer of unmatched processing power, 95% of which is presently being left unutilized. However, the global community appears to be still hesitant at tapping into the well of unharnessed potential offered by exploiting distributed computing. Reasons include the lack of personal incentive for participants, and the high degree of expertise required from application developers.

Our vision is to tackle the aforementioned obstacles by building a P2P system capable of deploying user-defined tasks onto the network for distributed execution. Users would only be expected to write standard concurrent code accessing our application programming interface, and may rely on the system to transparently provide for optimal task distribution, process migration, message delivery, global state, fault tolerance, and recovery. Strong mobility during process migration is achieved by pre-processing the source code. Our results indicate that near-linear efficiencies – approximately 94% ± 2% of the optimal – may be obtained for adequately coarse-grained applications, even when deployed on a heterogeneous network.

1. Introduction and Background

An enticing eventuality of any promising novelty is its evolution from an esoteric research area to an unbounded opportunity for the masses. Quintessential technological phenomena include the Internet enabling global connectivity [1], the World Wide Web providing us with a virtually-unlimited repository of information [2], and peer-to-peer (P2P) file-sharing applications facilitating the unhindered distribution of any form of media [3]. One area which, however, appears to be lagging behind is that of exploiting distributed computing power [4].

This is somewhat disconcerting, since the aggregation of typical home computers through a P2P framework over the Internet would yield a virtual supercomputer of unmatched processing power [5]. Furthermore, most of this power is presently being left unutilized; Schrage [6] and Davies [7] both report 95% CPU idle-time on typical powered workstations. If the surplus processing power of a group of computers could be pooled for consumption by users undergoing peak utilization periods, one would boost both throughput and efficiency, and mutual benefit would be derived by all participating parties [8].

Reasons which could account for the global community's hesitation at tapping into this well of unharnessed potential are various. Lack of personal incentive for participants could be an issue [5] – today's best-known distributed supercomputing applications, such as SETI@home, primarily benefit the developing entity rather than the participating home users. Many developers perceive a problem of accessibility, in that developing distributed applications often requires a high degree of expertise [9], rendering it an area exclusive to specialists. Furthermore, there is the inherent issue of having to provide for fault tolerance. Due to the dynamic nature of P2P networks, nodes may join or leave the network in an ad hoc manner, making it imperative for the application developer to implement redundancy and recovery measures [3] – a precaution which would be unnecessary on stable architectures, such as Grids [10].

1.1. Assimilation of Technologies

There are several fields of research which could give a positive contribution to the actualization of a distributed computing system. An eclectic assimilation of all such technologies would ensure a holistic treatment of the problem.

The **Grid** is a platform which aggregates heterogeneous collections of resources into a coherent infrastructure, affording users with reliable, cost-effective, and ready access to their capabilities [4]. Grid components are generally distributed (geographically dispersed), heterogeneous, and dynamic [11]. Nonetheless, such components may be accessed, individually or collectively, through a uniform interface, thereby abstracting away the infrastructural complexities to provide a virtual platform for the application developer [12].

Peer-to-peer (P2P) systems involve the dynamic establishment of decentralised network topologies by the participating nodes. Whilst resource-sharing is egalitarian in principle, node connectivity is generally ad hoc; thus, the system must be inherently capable of accommodating transient populations [13]. Loo [5] argues that, with the pervasion of high-end personal computers and high-bandwidth connections, the Internet's power is shifting back to its periphery; therefore, P2P is the "next logical step" (succeeding the client–server model) in the push towards maximising one's efficient usage of computational resources.

Autonomic computing entails the continuous observation and analysis of the (perceived) global state by the individual participants, and the reactive enactment of strategies by the said participants to address the changing environment – all in an automated manner. This allows the system to adapt rapidly to changes, and to exhibit resilience to adversities such as failing components, inaccurate resource advertisements, or unforeseen circumstances [14].

Parallelism and concurrency serve as the theoretical background to distributed systems, setting out the laws and promises that one may expect. Amdahl [15] contends that the speedup attainable by distributing a parallelizable problem over multiple processors would not be linear, but rather, an exponential approach towards a maximum value determined by the sequential portion of the problem. However, Gustafson [16] retorts that the relative size of the sequential portion of a problem can be reduced by expanding the parallelizable part – for example, by working at a greater precision or resolution. Communication overheads are gradually becoming less prevalent due to the fact that improvements in network performance are outpacing those of computational resources [17].

Process migration is the act of moving a process from one machine to another at runtime, preserving its execution state [18]. Strong mobility implies that the migration is transparent to the programmer. Low-level approaches for achieving strong mobility involve extending the operating system or virtual machine; high-level strategies entail altering some aspect of the compilation model, such as the original source code.

2. Aim and Objectives

Our aim is to design and implement a parallel processing framework which distributes user-defined tasks over a peer-to-peer network. Users would only be expected to write standard concurrent code accessing our application programming interface (API), and may rely on the system to transparently provide for optimal task distribution and fault tolerance guarantees, thereby addressing the issues discussed in the previous section.

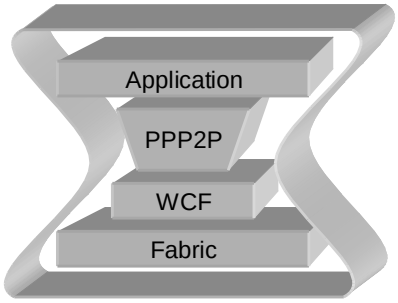

Figure 1. Our system (PPP2P) fits into the hourglass model, which is also used for Grids [17]. Connectivity is provided by the Windows Communication Foundation (WCF).

2.1. Objectives

The aim is concretely realized through the following objectives:

Concurrency: The system distributes each task's subtasks over several machines for simultaneous execution. Conversely, individual participant nodes are capable of executing multiple subtasks (possibly from different tasks) at the same time through multithreading.

Decentralisation: The structure of centralized topologies yields "inefficiencies, bottlenecks, and wasted resources" [3]. A fully-decentralized system divides the cost of ownership fairly amongst all participants, and avoids any central points of failure.

Scalability: The system may accommodate any number of participant nodes without experiencing any significant performance degradation. The throughput of the system (in terms of the number of tasks it can complete in a given time interval) scales almost linearly with the number (and processing power) of participating nodes available.

Efficiency: The system embodies a degree of intelligence for distributing tasks in an optimal manner. Each node maintains a partial or complete awareness of the composition of the global mesh, and delegates subtasks in a way that reduces their execution times and promotes overall throughput.

Fault tolerance: Since a P2P system involves nodes leaving the network in an unannounced and nondeterministic manner, recovery mechanisms are enacted for resuming subtasks which had been delegated to the departed nodes. This is achieved through a checkpointing strategy.

Ease-of-use: The API is aspired to be appealing, user-friendly, and unobtrusive, to the extent that it could be easily utilised without requiring technical knowledge of the inner workings of the system.

2.2. Features

The following set of features (which is accessible through our API) spans a wide range of programming models, allowing the application developers to pick the subset which best suits their needs or methodology:

- The initialization and management of **virtual computers** which enrol the requested number of workers to participate in a particular task.
- The management and deployment of **virtual threads** through virtual computers, with each virtual thread encapsulating the execution of a specific subtask.
- The **returning of results** to the user application once a virtual thread completes execution. (Note that this feature, along with the previous two, would suffice for a task-farming application.)
- The **reporting of *partial* results** to the user application during the virtual thread's execution, using an event-driven approach.
- Direct communication between any pair of virtual threads using buffered **message passing**.
- Mandatorily-synchronized access to **global variables**, whose values would be common to all virtual threads within a virtual computer.
- **Checkpointing** of the execution state of each individual virtual thread (using process migration), allowing it to be resumed on another peer if its host should disconnect or die.

3. Design

Our design strategy is aligned with the recent wave promoting the coalescence of Grid and P2P technologies [13], [19]. We provide a uniform interface for the exploitation of heterogeneous resources by independently-developed user applications (as in Grids), but support an egalitarian and ad hoc participation model (as in P2P). The stability of the infrastructure would be maintained dynamically using concepts borrowed from autonomic computing.

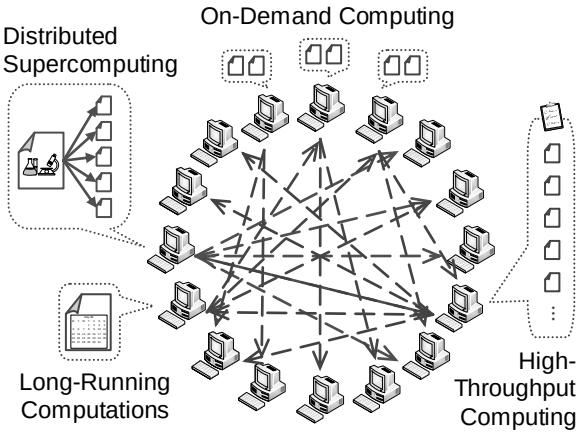

Figure 2. The various usage scenarios under which our system may be employed, possibly simultaneously.

3.1. Duality

The system is designed to effectuate an interplay between two distinct but closely-interdependent projects. The **Peer Controller**, on one hand, handles the local peer's participation in the P2P network. It assumes responsibility for deploying tasks submitted by local clients (user applications), as well as contributing to the execution of tasks from other nodes' clients (submitted through their respective Peer Controllers). The **Leverage** library, on the other hand, integrates with the user application, allowing it to utilize the functionality of the Peer Controller and, subsequently, the rest of the system.

This arrangement implies that the system may be perceived through two different viewpoints: the structural architecture of the Peer Controller, and the virtualisation platform given by the Leverage library. The functionality of the latter is enabled through the invocation of the former; however, it is exposed to the user at a substantial level of virtualisation (abstraction), since several mechanisms – such as task distribution, migration, message delivery, global state, fault tolerance, and recovery – would be provided by the underlying system.

3.2. Fault Tolerance

The **Pre-Processor** is a passive component which prepares an application for strong mobility. Intended to be run *before* compiling the user application, it converts the source code to a version which explicitly maintains the execution state, but preserves the semantics of the original code.

At periodic intervals, each virtual thread would checkpoint itself by anticipatively migrating a copy of

its execution state to a remote node, where it is held in dormancy. Should the node executing the virtual thread undergo an unannounced departure, the virtual thread would be resumed, from its last checkpointed state, on another node. This is all performed transparently to the user application, which would not even be aware of the node failure or migration.

4. Implementation

4.1. Peer Controller

Most of the mechanisms required for achieving the specified feature set are incorporated into the Peer Controller – the Leverage library only provides the access to these mechanisms. The Peer Controller is organized into an assemblage of functionally-cohesive components, each being responsible for some specific aspect of its activities or behaviours.

The root of the composition hierarchy is, expectedly, the **peer controller** component. The **participation manager** has a twofold responsibility: periodically announcing the peer's participation in the global mesh, and maintaining a (partial or complete) record of the other participating peers.

The **deployer** initiates and manages a virtual computer for a local leveraged client. The **worker**, on the other hand, enrols in a virtual computer established by a remote peer. It instantiates an **executor** for executing each virtual thread delegated to it by the remote deployer. **Peer unit proxies** are responsible for establishing any direct connections required between peer units (namely, deployers and workers) for unicast communication. They provide a layer of location transparency for the virtual threads, allowing them to intercommunicate without needing to know whether the target virtual thread is being executed in the same worker or in a remote one.

The **assembly manager** is responsible for retrieving, caching, and loading assemblies. The **address resolver** is consulted by the worker proxy during message passing; it resolves the virtual thread ID of the target virtual thread to the address of the (local or remote) executor on which it is presently deployed. The **message manager** handles the buffered transfer of messages between virtual threads. The **global variable manager** coordinates synchronized access to global variables. The **worker quota maintainer** ensures that the quota for the requested number of workers to enrol in the virtual computer is maintained throughout its lifetime. The **enrolment requestor** issues worker enrolment requests to available peers concurrently in order to meet the quota for the requested number of workers. Enrolment requestors are invoked either upon virtual computer initialization, or when the number of enrolled workers falls below the quota (because of node failures). Finally, the **virtual thread delegator** distributes virtual threads to the enrolled workers for execution.

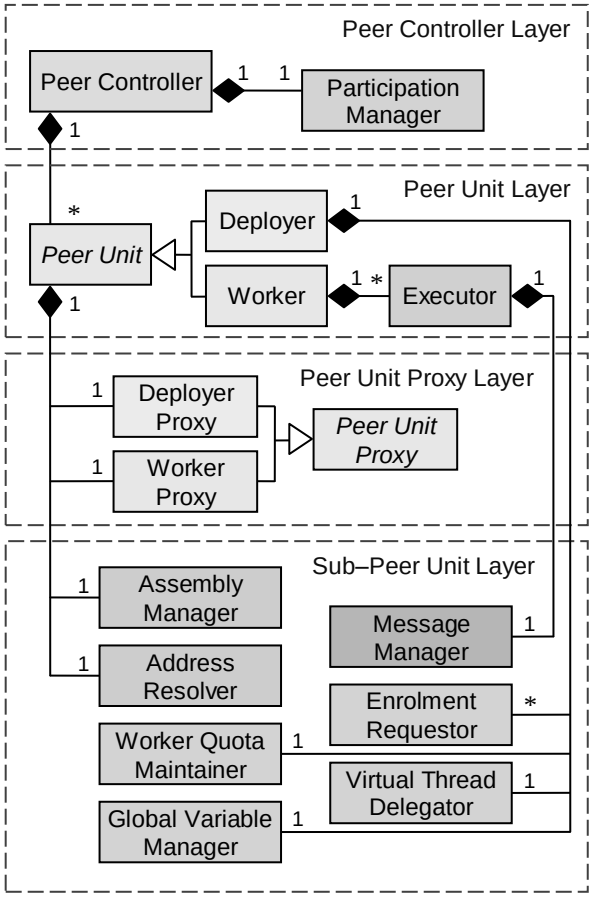

Figure 3. UML 2 class diagram for the structural architecture of the Peer Controller.

4.2. Pre-Processor

The Pre-Processor prepares the source code for strong mobility by subjecting it to a conversion sequence. It starts off by reading and parsing the source code, creating an abstract syntax tree (AST). Next, it passes the AST through a pipeline of visitors and transformers.

Methods which may, directly or indirectly, invoke a checkpoint are encapsulated within inner classes; each inner class would represent the particular method's stack frame. The body of each checkpointable method is converted into a series of logical blocks, with each logical block being associated with a particular value of an artificial program counter. Parameters and local variables within the checkpointable methods are promoted to instance variables (fields)

within the encapsulating inner class, allowing their value to be serialized when a checkpoint is performed.

Finally, the contents of the converted AST are output back to a source code string, which may then be compiled into an ordinary .NET assembly.

5. Evaluation

To evaluate the capabilities of our system quantitatively, we developed a parallelizable version of a genetic algorithm for the travelling salesman problem. We devised a scheme whereby each virtual thread would evolve its own population, yet interbreed its best chromosomes with the other virtual threads at regular intervals in order to promote convergence. Each virtual thread was deployed onto a dedicated peer to facilitate evaluation.

Prevailing trends in Grid and P2P systems embrace participant heterogeneity congenitally [11], [13]. Therefore, we opted to run our tests on heterogeneous setups, using the definitions for speedup presented by Donaldson et al. [20], among others. The processing rate of a problem on a machine or heterogeneous system, R_{M_i} or R_H, is obtained by dividing the problem size by its execution time on the said machine or system. The attained speedup of a heterogeneous system, S_H, is defined as the ratio of the processing rate of the heterogeneous system, R_H, to the processing rate of the fastest participant, R_{M_1}. Barring the occurrence of super-linear phenomena [21], the maximum possible speedup that may be expected from the heterogeneous system, $S_{H,max}$, is the ratio of the sum of the processing rates of the individual participating machines, $\sum_i^n R_{M_i}$, to the processing rate of the fastest participant, R_{M_1}. Finally, the efficiency of a heterogeneous system, η_H, is intuitively defined as the ratio of the attained speedup to the maximum speedup.

$$S_H = \frac{R_H}{R_{M_1}} \qquad S_{H,max} = \frac{\sum_i^n R_{M_i}}{R_{M_1}} \qquad \eta_H = \frac{S_H}{S_{H,max}}$$

To measure the processing rates of the individual machines, we executed a single virtual thread separately on each machine, without making use of the distributed processing infrastructure. This allowed us to obtain a close estimate of what would have been the performance of the sequential version of the algorithm. Subsequently, we arranged our available machines into a number of heterogeneous setups and ran another series of tests, this time deploying the algorithm on all available peers. Speedups and efficiencies were calculated using the formulae presented above.

As one may observe from Figure 4, the system incurs substantial initialization costs; however, these gradually level out with larger problem sizes. For our largest problem size, efficiencies were 95.7%, 94.2%, and 93.0% for setups containing 2, 3, and 4 machines, respectively. This performance is considered quite acceptable for a communicative algorithm, where efficiencies exceeding 90% may be classified as near-linear [21]. The observation that efficiency diminishes with increasing levels of concurrency conforms to Amdahl's Law [15]; in our case, the sequential part was the interbreeding, which was performed through a mandatorily-synchronized global variable.

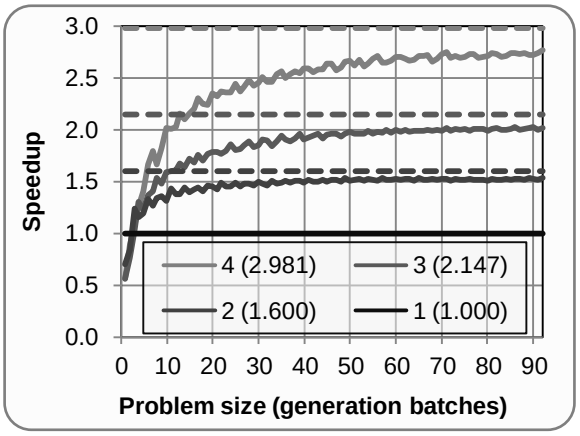

Figure 4. Speedups attained for different problem sizes, under three heterogeneous setups (plus the reference setup). The legend gives the number of peers in each setup. Maximum speedup is indicated through dashed lines within the graph, and in parentheses within the legend.

6. Conclusions

Distributed computing has yet to break into its synergy phase, possibly due to the socio-institutional inertia which prohibits mainstream adoption of a new paradigm until it has reached a certain critical mass [22]. There have been various initiatives intended to encourage a wider adoption of distributed computing, ranging from humble prototypes, such as our own, to full-fledged Grid dissemination projects, such as EUMEDGRID. However, what the world appears to be waiting for is a killer application which would open up this unharnessed realm to the global community at large, just as the Mosaic browser had done for the Web, and Napster for P2P file-sharing. The procurement of such an application would be a prodigious and gratifying feat which would propel a new era of innovation.

Cycle-harvesting through distributed systems has been advocated for several years. Back in 1995, Anderson et al. [8] had declared that networks of workstations (NOWs) would eventually become the primary computing infrastructure, amassing the ca-

pacities of the workstations' memory, storage, and processing units, to serve "*all* the needs of computer users" [emphasis in original]. Both Loo [5] and Davies [7] postulate that the ultimate manifestation of this trend would be the emergence of a global platform which leverages the ubiquity of the Internet to aggregate processing power from computers all over the world, joining the ranks of quintessential technological phenomena. The pursuing spirit of utilitarianism would effectively establish – as eloquently dubbed by Erlanger [23] – "the poor man's supercomputer".

References

[1] L. Kleinrock, "An Internet vision: the invisible global infrastructure," *Ad Hoc Networks*, vol. 1, 2003, pp. 3–11.

[2] S. Brin, "Extracting Patterns and Relations from the World Wide Web," *WebDB '98: Selected papers from the International Workshop on The World Wide Web and Databases*, London, 1999, pp. 172–183.

[3] D.S. Milojičić, et al., "Peer-to-Peer Computing," Technical Report HPL-2002-57R1, HP Laboratories, Palo Alto, California, 2002.

[4] I. Foster and C. Kesselman, "Computational Grids," *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds., Morgan Kaufmann Publishers, San Francisco, 1998, ch. 2, pp. 15–52.

[5] A.W. Loo, "The Future of Peer-to-Peer Computing," *Communications of the ACM*, vol. 46, no. 9, Sep. 2003, pp. 56–61.

[6] M. Schrage, "Piranha processing – utilizing your down time," *HPCwire (Electronic Newsletter)*, Aug. 1992.

[7] A. Davies, "Computational intermediation and the evolution of computation as a commodity," *Applied Economics*, vol. 36, no. 11, June 2004, pp. 1131–1142.

[8] T.E. Anderson, D.E. Culler, and D.A. Patterson, "A Case for NOW (Networks of Workstations)," *IEEE Micro*, vol. 15, no. 1, Feb. 1995, pp. 54–64.

[9] J. Waldo, G. Wyant, A. Wollrath, and S.C. Kendall, "A Note on Distributed Computing," *Selected Presentations and Invited Papers, Second International Workshop on Mobile Object Systems (MOS '96) – Towards the Programmable Internet*, London, 1996, pp. 49–64.

[10] D. De Roure, M.A. Baker, N.R. Jennings, and N.R. Shadbolt, "The evolution of the Grid," *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A.J.G. Hey, eds., John Wiley & Sons, Chichester, 2003, ch. 3, pp. 65–100.

[11] F. Berman, G. Fox, and A.J.G. Hey, "The Grid: Past, Present, Future," *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A.J.G. Hey, eds., John Wiley & Sons, Chichester, 2003, ch. 1, pp. 9–50.

[12] H. Stockinger, "Defining the grid: a snapshot on the current view," *The Journal of Supercomputing*, vol. 42, no. 1, Oct. 2007, pp. 3–17.

[13] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys*, vol. 36, no. 4, Dec. 2004, pp. 335–371.

[14] P. Pattnaik, K. Ekanadham, and J. Jann, "Autonomic Computing and Grid," *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A.J.G. Hey, eds., John Wiley & Sons, Chichester, 2003, ch. 13, pp. 351–361.

[15] G.M. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," *AFIPS Conference Proceedings*, AFIPS Press, vol. 30, Apr. 1967, pp. 483–485.

[16] J.L. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, vol. 31, no. 5, May 1988, pp. 532–533.

[17] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A.J.G. Hey, eds., John Wiley & Sons, Chichester, 2003, ch. 2, pp. 51–63.

[18] D.S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process Migration," *ACM Computing Surveys*, vol. 32, no. 3, Sep. 2000, pp. 241–299.

[19] I. Foster and A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing," *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, 2003, pp. 118–128.

[20] V. Donaldson, F. Berman, and R. Paturi, "Program Speedup in a Heterogeneous Computing Network," *Journal of Parallel and Distributed Computing*, vol. 21, no. 3, June 1994, pp. 316–322.

[21] E. Alba, A.J. Nebro, and J.M. Troya, "Heterogeneous Computing and Parallel Genetic Algorithms," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, 2002, pp. 1362–1385.

[22] C. Perez, "Technological Revolutions, Paradigm Shifts and Socio-Institutional Change," *Globalization, Economic Development and Inequality: An Alternative Perspective*, E. Reinert, ed., Edward Elgar, Cheltenham, 2004, pp. 217–242.

[23] L. Erlanger, "Distributed Computing: An Introduction", Apr. 2002; http://www.extremetech.com/article2/0,1697,1154105,00.asp.