

Scalability Dilemma and Statistic Multiplexed Computing -- A Theory and Experiment

Justin Yuan Shi*, Yasin Celik

Department of Computer and Information Sciences, Temple University, Pennsylvania 19122, USA

ARTICLE INFO

Article history:

Received: 21 May, 2017

Accepted: 20 July, 2017

Online: 10 August, 2017

Keywords:

Extreme Scale Computing

Statistic Multiplexed Computing

ABSTRACT

The For the last three decades, end-to-end computing paradigms, such as MPI (Message Passing Interface), RPC (Remote Procedure Call) and RMI (Remote Method Invocation), have been the de facto paradigms for distributed and parallel programming. Despite of the successes, applications built using these paradigms suffer due to the proportionality factor of crash in the application with its size. Checkpoint/restore and backup/recovery are the only means to save otherwise lost critical information. The scalability dilemma is such a practical challenge that the probability of the data losses increases as the application scales in size. The theoretical significance of this practical challenge is that it undermines the fundamental structure of the scientific discovery process and mission critical services in production today.

In 1997, the direct use of end-to-end reference model in distributed programming was recognized as a fallacy. The scalability dilemma was predicted. However, this voice was overrun by the passage of time. Today, the rapidly growing digitized data demands solving the increasingly critical scalability challenges. Computing architecture scalability, although loosely defined, is now the front and center of large-scale computing efforts.

Constrained only by the economic law of diminishing returns, this paper proposes a narrow definition of a Scalable Computing Service (SCS). Three scalability tests are also proposed in order to distinguish service architecture flaws from poor application programming. Scalable data intensive service requires additional treatments. Thus, the data storage is assumed reliable in this paper. A single-sided Statistic Multiplexed Computing (SMC) paradigm is proposed. A UVR (Unidirectional Virtual Ring) SMC architecture is examined under SCS tests. SMC was designed to circumvent the well-known impossibility of end-to-end paradigms. It relies on the proven statistic multiplexing principle to deliver reliable service using faulty components as the infrastructure expands or contracts.

To demonstrate the feasibility of such a theoretical SCS, an organized suite of experiments were conducted comparing two SMC prototypes against MPI (Message Passing Interface) using a naive dense matrix multiplication application. Consistently better SMC performance results are reported.

1. Introduction

For the last three decades, end-to-end computing has been the de facto paradigm for distributed and parallel programming. MPI (Message Passing Interface), RPC (Remote Procedure Call), OpenMP (share memory) and RMI (Remote Method Invocation) are all end-to-end programming paradigms. A myth persisted since the 1990's that while the data communication community is well served by the end-to-end paradigm, in 1997, direct use of the end-to-end protocol in distributed computing was cited as a fallacy [1]. The computing service scalability dilemma was predicted.

*Corresponding Author: Justin Yuan Shi, Science Education Research Center 310, Temple University, +1 205-204-6437, USA | Email: shi@temple.edu

A reexamination of the 1993 proof of the impossibility of implementing reliable communication in the face of crashes [2] exposed the root cause of the alleged fallacy: the robust communication protocols are ineffective when either the sender or the receiver in the end-to-end communication could crash arbitrarily. No error detection and recovery methods can reverse this impossibility.

Direct use of end-to-end reference model in computing applications leaves massively many potential end-point crashes when the application runs (Figure 1). Although the probability of each end-point crash is very small, the growth in application size (thus the processing infrastructure) is guaranteed to increase the planned and unplanned service down times and data loss probabilities. The reproducibility of a large scale application

(service) becomes increasingly more difficult [3]. These risks are tolerable when the computing clusters are small. They have become increasingly unbearable as the computing services facing increasingly expanding loads.

Incidentally, reliable communication is possible using faulty components. This was also proved in early 1980's [4]. There are three assumptions in the proof: a) all data packets must decoupled from transmission devices, b) both the sender and the receiver must be reliable, and c) there are infinite supplies of resources. In literature, this is often called the "best effort" service. Theoretically, the best effort communication guarantees 100% reliability as long as the infrastructure affords the minimal survivable resource set at the time of needs. This was the theoretical basis of the OSI (Open System Interconnection) reference model [5].

It then follows that the "best effort" computing should also be possible by removing the reliable receiver assumption in the basic program-program communication protocols. In other words, while it is reasonable for a data communication application to assume reliable sender and receiver at the time of need, it is not reasonable to assume reliable receivers in any clustered computing services, since the growing service infrastructure will make it increasingly unlikely.

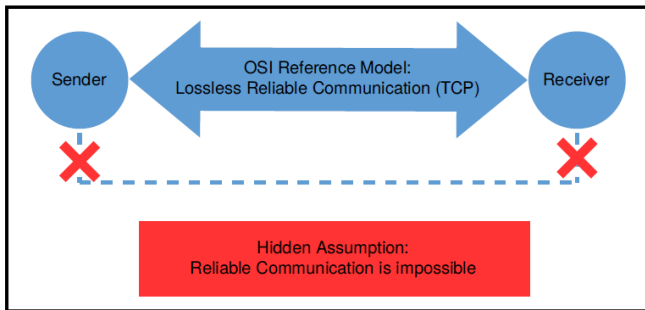


Figure 1. OSI Standards and Its Hidden Assumption

This has led to the design of a single-sided parallel computing paradigm. The new paradigm must multiplex all resources or the application fault tolerance is not attainable (Figure 2). This framework is called Statistic Multiplexed Computing (SMC) or Stateless Parallel Processing. Today, as the single processor speed approaches to a plateau, it seems that unconstrained clustered computing is the only likely future computing architecture. No single big-CPU machine, even if quantum class machines, would be able to meet the exponentially growing data processing needs.

This paper is organized as follows. Section 2 introduces the assumptions and a narrow definition of a Scalable Computing Service (SCS). It also includes three "Reproducible Architecture Tests" as the necessary and sufficient conditions for SCS. In this paper, the data storage is considered stable and lossless. Data intensive SCS will need additional requirements. A single-sided Statistic Multiplexed Computing (SMC) architecture is presented. Section 3 provides a high level examination of the single-sided SMC or Stateless Parallel Processing (SPP) paradigm using the proposed scalability tests. Section 4 illustrates the need for granularity tuning for optimal parallel processing. Section 5 evaluates MPI, Hadoop, Spark and two SMC prototypes: Synergy (SPP) and AnkaCom (SMC) using the scalability tests. Section 6 documents the experiment design and objectives. Section 7 reports the computational results. Section 8 contains the summary.

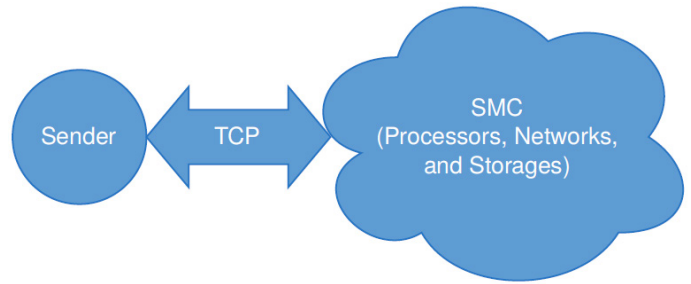


Figure 2. Single-Sided SMC Paradigm

2. Assumption, Definition and Tests

Scalability is the capability of a system to handle a growing amount of work. It is well understood in an economic context, where a company's scalability undermines the potentials economic growth of the company. Since the future societal economic growth depends on all available computing services, the importance of the computing service scalability is self-evident.

The economic law of diminishing returns states that in all productive processes, adding more of one factor of production, while holding all others constant, will at some point yield lower incremental per-unit returns. Parallel computing using multiple processors and networks obey the same law.

There are many possible dimensions in scalability measures. For compute intensive services, assuming infinite supplies of processors and networks, the scalable computing research challenge is to devise a computing service architecture that can scale indefinitely in order to meet the growing demands. The Internet and SMP (Symmetric Multiprocessing) systems are such service architectures. Delivering robust service for growing demands, however, has met the seemingly insurmountable challenge of the scalability dilemma. The infinite resource assumption was rooted in practice. This assumption also makes the theoretical discussions possible.

The dependency on reliable processors and networks for delivering reliable service is the root cause of the scalability dilemma. Since electronic components can suffer random failures, a robust computing architecture must be able to exploit all useable components at the time of need. Thus the *infinite resource assumption* can be translated into the *minimal survivable resource set* assumption in practice. That is, all discussions on scalable performance, reliability and service quality are based on the assumption that the processing architecture affords the "minimal survivable resource set" at the time of need. The architecture design challenge is to build a mechanical structure capable of leveraging all available resources at the time of need without reprogramming the application.

In this paper, we also assume that the storage is reliable. Data intensive SCS requires additional treatments following the same principles [6].

Definition: A Scalable Computing Service (SCS) is a computing service that allows unlimited infrastructure expansion without reliability and service quality degradation. Application performance should subject to the same definition until the law of diminishing returns applies.

Parallel applications are the integral parts of the scientific discovery processes. It is thus important to ensure that extreme

scale service architecture does not hinder the scientific discovery process. The service reproducibility requirement is implied. Further, it would be difficult to distinguish a computing architecture design flaw from a poorly composed non-scalable application (crossing the point of diminishing returns prematurely), the following reproducible SCS tests are proposed:

- a. **Share-nothing Test.** No processors or networks can be assumed reliable while still delivering full range of services. Not passing this test breaks the scalable reliability definition.
- b. **Sublinear Cost Test.** The runtime management overhead should be bound within a sublinear factor with regard to the infrastructure size. Failing this test means diminishing benefits when the application up-scales. It breaks the scalable performance definition.
- c. **Reproducible Node Test.** Each node in the architecture must be able to reproduce identical semantically acceptable results given identical inputs. These include both deterministic and non-deterministic programs. Failing this test will break the service quality definition.

Test (a) requires a single-sided programming paradigm (such as Figure 2) and a multiplexing runtime architecture. Tests (a) and (b) are the necessary and sufficient conditions for SCS. Tests (a), (b) and (c) are the necessary and sufficient conditions for a reproducible SCS, since the infrastructure ensures reliable services in any scale using faulty components.

All modern computers running reasonable software can pass the *Reproducible Node Test*. The *Share-nothing* and *Sublinear Cost Tests* are more difficult.

Services built using the end-to-end reference model fail the *Share-nothing Test*. The end-to-end reference model requires explicit receiver addresses in an IP-addressable network. This seemingly innocent requirement causes the entire application to depend on the reliability of all processors and networks. This is the root cause of the scalability dilemma [1].

The following computing systems can pass the *Share-nothing Test*: a) Dataflow machines [7], b) Tuple Space machines [8], and c) services by Content Addressable Networks [9]. Similarly, the search engines and SMP (Symmetric Multiprocessing) systems also qualify. These systems have two common features: a) API (Application Programming Interface) has no fixed destination (single-sided), and b) every task can be processed by any processing elements.

Dataflow and Tuple Space machines are elegant automated parallel systems but suffer the poor performance stigma. Content Addressable Networks are designed for the next-generation Internet services. It is not typically used for parallel computing. Search engines and SMP systems are designed to service multiple clients effectively. They are ineffective for solving the single application's scalability dilemma.

However, putting these systems together brought a unique opportunity: Stateless Parallel Processing (SPP) or Statistic Multiplexed Computing (SMC) [10]-[15]. Specifically, it is possible to build a "Service Content Addressable Network" using the Tuple Space semantics and dataflow principle thus making the "best effort computing" practical.

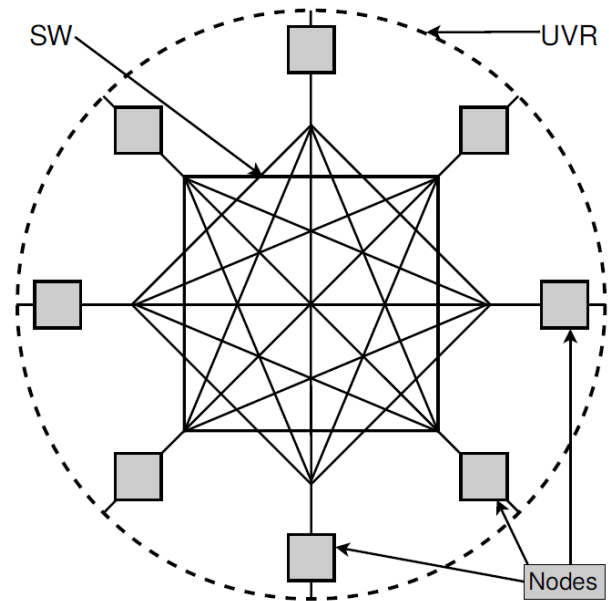


Figure 3. Statistic Multiplexed Computing Architecture

Figure 3 illustrates such a multiplexing computing service under a UVR (Unidirectional Virtual Ring) architecture.

3. Single-sided Statistic Multiplexing

Passing the *Share-nothing Test* for a computing architecture requires the absence of all component reliability assumptions (except for the storage for this paper). The service communication architecture must be capable of exploiting all possible network topologies. In Figure 3, SW is a collection of network switches in any topology. Each node is a standalone computer with any number of cores, local memory, storage and multiple network interfaces. UVR (Unidirectional Virtual Ring) implements a Service Content Addressable Network (or Tuple Switching Network (TSN) [16]) using all available processing and networking components. This allows zero single point failures regardless the number of networks and processors for a given application.

The UVR architecture ensures that except for the Master (the client of the parallel computing service), there is no need for explicit IP addresses for data exchanges.

Unlike traditional parallel Masters, the SMC Master is part of UVR architecture in that it is responsible for tuple retransmission discipline and redundancy handling. Actual data exchanges are implemented using all available network links directly. Services built using the TSN do not subject to the end-to-end impossibility. The high level UVR computing concept passes the *Share-nothing Test*. The "best effort computing" idea could become feasible, if the implementation also passes the same test.

Once the application runs, the dataflow semantics allow the computing infrastructure to automatically form SIMD (Single Instruction Multiple Data), MIMD (Multiple Instruction Multiple Data) and pipeline clusters at runtime. These effects are identical to early dataflow machines [7].

However, passing the *Sublinear Cost Test* needs more work. On the surface, it seems impossible to traverse P nodes paying less than linear traversal cost. However, k -order multicast can cut the

UVR traversal complexity to $O(\lg_k P)$ [17]. Further, once a tuple is matched against its processor, actual data exchange will be done directly. Imposing an order on the tuple values can further force the tuple matching overheads to $O(1)$ using the ideas from Consistent Hashing [18]. For practical applications, the one-time $O(\lg_k P)$ traversal cost seems reasonable for deploying millions of nodes.

4. Single-sided Statistic Multiplexing

All compute intensive applications are typically partitioned into parallel tasks that can be processed using SIMD, MIMD and pipeline parallelisms. Since data exchange costs time, the partition size or *processing granularity* determines the ultimately deliverable performance by the service architecture. The best performance is delivered when all tasks terminate at the exactly the same time. A small granularity allows for higher concurrency but at the expense of higher data exchange costs. A bigger granularity risks lower concurrency and longer synchronization times due to differences in processing capabilities. The optimal granularity delivers the best possible parallel performance for the application running on a given processing environment.

The optimal *processing granularity* (G) of an application defines the Termination Time Equilibrium [19], [20] for that application. Finding the optimal G will allow the SMC applications to overcome the poor performance stigma of earlier dataflow and Tuple Space machines.

The principle of finding the optimal G is analogous to the use of Snell's Law for solving the Brachistochrone problem in physics and mathematics [21]. Figure 4 illustrates that the solution to the fastest descent under the influence of uniform gravitational field

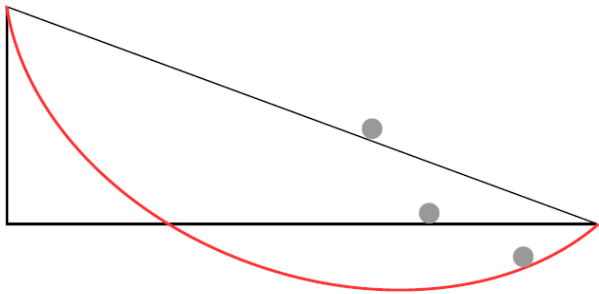


Figure 4. The Brachistochrone Problem.

and friction-less surfaces is not the straight line that connects the source and the destination, but the Brachistochrone curve that optimizes the gravitational and normal forces. It is a cycloid. Using fixed application partitioning is like the straight line (which happened to be the slowest descent). The astonishing feature of the Brachistochrone curve is that it is also called "Tautochrone" that regardless where you start on the curve, all will reach the destination at the same time. In parallel processing, the Brachistochrone curve represents the optimal G 's where the overall computing time equals to the overall data exchange time. As will be shown, if we tune G carefully, there are indeed multiple optimal G 's (modulated under a communication overhead curve) for every compute intensive kernel [19].

Further, the optimal G is only discoverable without reprogramming under the single-sided paradigms. For end-to-end computing programs, unless the task distribution is pooled [22] or

using the single-sided "scatter_v" call in MPI, once compiled, changing processing granularity requires reprogramming.

Other single-sided systems include Hadoop [23] and Spark [24]. They have demonstrated substantially better reliability and performance than similar end-to-end distributed computing systems. But performance against bare metal HPC programs have not being rigorously investigated.

The remaining of this paper reports computation experiments comparing two SMC prototypes: Synergy and AnkaCom against MPI (Message Passing Interface). The Brachistochrone effects are also demonstrated.

5. Computing Service Architectures and Scalability Tests

The MPI parallel processing architecture is embedded in its application programs. The (node) operating systems are responsible for the basic data communication and task execution functions, the programmer has explicit controls of the parallel machines. The explicit end-point requirement makes it impossible to pass the *share-nothing test*. Even with a "task pool" implementation, unless the implementation is completely distributed, thus forming a "application content addressable network", passing the *share-nothing test* remains negative.

The Hadoop system uses the single-sided $\langle key, value \rangle$ API (Application Programming Interface) for reliable large scale distributed processing. It can pass the *share-nothing test* at the concept level. Unfortunately, it fails the same test at the implementation level due to the use of the RPC (Remote Procedure Call) protocol. This results in the "single namenode architecture". The "namenode" is the single-point failure of the entire system. However, even though Hadoop cannot pass the *Share-nothing Test*, its single-sided API allowed much better runtime fault tolerance than other systems. Many very large scale successful experiments are completed within the 100M file design limitation.

The Spark system relies on the Hadoop File System (HDFS) but leverages high speed in-memory processing. Its scalability test is identical to Hadoop.

The Ethereum project [25] is a decentralized distributed computing platform that runs smart contracts: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference. It can pass the *share-nothing test* easily. These applications run on a custom built blockchain, a shared global infrastructure that can move value around and represent the ownership of property. This enables developers to create markets, store registries of debts or promises, move funds in accordance with instructions given long in the past (like a will or a futures contract) and many other things that have not been invented yet, all without a middle man or counterparty risk. However, its Blockchain implementation cannot pass the *Sublinear Cost Test* due to linear overhead increases when the chain expands.

The Synergy system is also a distributed parallel computing system that uses named Tuple Space Servers to simulate the TSN. Although it can pass the *Share-noting Test* at the concept level, the actual implementation fails the test due to the use of the named Tuple Space Servers with fixed IP addresses [9].

The AnkaCom system [26] is a fully distributed peer-to-peer TSN implementation. It can pass the *share-nothing test* at the concept level and the implementation level. The fundamental

departure point of the AnkaCom implementation from others is the absence of the single-ACK/NAK assumption as compared to the RPC protocol. It can also pass the *Sublinear cost test* by implementing the one-time $O(\lg_k P)$ UVR traversal protocol.

6. Experiment Design

Matrix multiplication is frequently used in scientific simulations and engineering applications. Dense matrix multiplication has high computing and communication requirements that can saturate resources quickly. It is selected as the benchmark for this study. Matrix multiplication is a “regular” parallel applications that fixed application partition seems reasonable for using a dedicated cluster of bare metal processors. The MPI parallel matrix multiplication program is implemented in C. It is optimized for locality for the three nested (i,j,k) loops. The Synergy program is also implemented in C and similarly optimized. The AnkaCom program is implemented in Java. The loop order is also optimized.

The MPI and Synergy runtime systems are implemented in C. The AnkaCom runtime is implemented in Java.

The computing platforms include the NSF (National Science Foundation) Chameleon bare metal cluster [27] at TACC (Texas Advanced Computing Center) and the owlsnest.hpc.temple.edu traditional bare metal cluster [28] at Temple University. The benchmark application is a naive dense square matrix multiplication application. The SMC prototypes are Synergy 3.0+ and AnkaCom 1.0. OpenMPI versions 1.4.4 and 1.10.0 are used in the experiments.

- *Owlsnest cluster* has multiple IB (Infiniband) support. The Chameleon cluster only has single IB support at the time of experiments.}
- The *Chameleon* and *Owlsnest* bare metal clusters have same number of cores (48) and sufficient memory for many-core experiments (256 GB for Chameleon) per node and (543GB for Owlsnest).

The Synergy implementation will only sustain to a small number of cores before saturating the single-threaded Tuple Space server. Thus, every test using large number of cores is also a sense of reliability test of the UVR concept.

The AnkaCom implementation will sustain to any number of nodes and cores due to the implementation of distributed TSN. The Java runtime overheads should be much higher than the C-MPI combination. For AnkaCom, the IB support comes from the built-in Java library. The C-MPI implementation has a custom IB driver.

There are three groups of experiments:

- *Single-Node Single-Core*. This group of tests examines the runtime’s ability to leverage multiple hardware components in parallel by overlapping single-core computing, communication and disk activities.
- *Single-Node Multiple-Core*. This group of tests examines the parallel runtime system’s ability to use up to 48 cores concurrently in addition to other communication and storage components.
- *Multiple-Node Multiple-Core*. This group of tests examines the parallel runtime system’s ability to harness multiple multicore nodes effectively. In this

study, we used twenty 12 core nodes on Owlsnest with dual IB support.

The single-sided API affords SMC applications a flexible parallel task pooling capability that is not supported by direct message passing systems such as MPI. The SMC workers can be programmed to compute for tasks of different sizes without re-programming. In contrast, the MPI applications rely on fixed N/P partitioning.

7. Computational Results

7.1. Single-Node Single-Core Results

. The Java compiler does not have an optimization option.

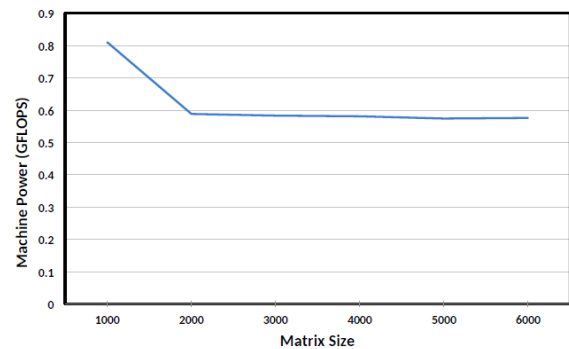


Figure 5. Single-Node Single-Core (Sequential)

The baseline experiment is provided by a sequential C-program running on a single core. Figure 6 shows the results in GLOPS on Owlsnest running matrix of sizes 1000 - 6000.

The significance of the Figure 5 curve is that it depicts the typical “Cache, Memory, Swap and Die” (CMSD) single core performance behavior. Understanding this behavior enables further quantitative application scalability analysis [19].

A partitioned parallel program running on a single node with a single core will exhibit different behavior than a sequential program running in the same environment. The partitioned programs can exploit hidden concurrencies even in the single node single core environment. Figure 6 reports the single-core single worker tests for both MPI and Synergy against the sequential program (in yellow). Due to the restriction of end-to-end communication, the MPI (Open MPI 1.4.4) program was not able

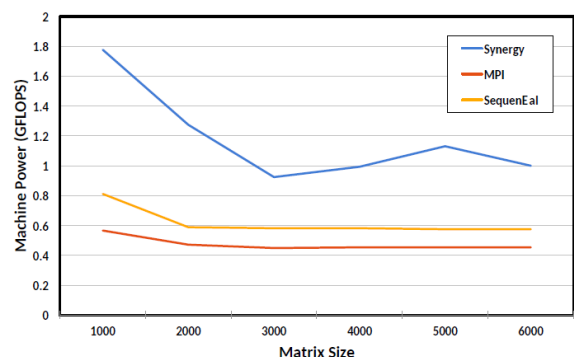


Figure 6. Single-Node Single-Core (Parallel)

to compete against the sequential code in different matrix sizes (1000-6000) consistently. The Chameleon results are similar.

7.2. Single-Node Multiple-Core Results

Single node multi-core is best suited for MPI and Synergy architectures since the inter-program data exchanges can be very fast. Figure 7 reports the granularity tuning results for MPI and Synergy on Chameleon for $N=9000, P=45$ (factor of 9000). In Figure 7, the optimal granularity sizes are: 11, 22, and 33. The relationship between these sizes are given by the volatility power indices of the 45 cores at their peaks: 303, 409, and 818 [20]. At these optimal points, the synchronization overhead is near zero (note the Brachistochrone analogy). Further quantitative scalability analysis becomes possible using the optimized performance and simpler time complexity models [20]. The MPI (Open MPI 1.10.0) on Centos 7 delivered consistently worse performance than the worst tuned Synergy performance (5.1GFLOPS).

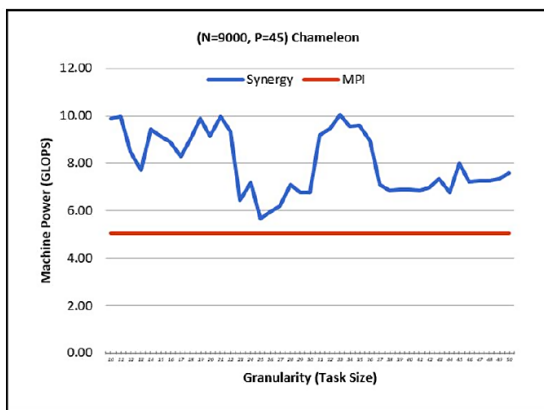


Figure 7. Single-Node Multiple-Core Granularity Tuned

Figure 8 reports a broader range of tests for different matrix sizes without granularity tuning ($G=20$ fixed). Synergy still outperforms MPI when the matrix size exceeds 3,200. These results are consistent with Figure 6: the bigger the problem sizes, the better the Synergy performance.

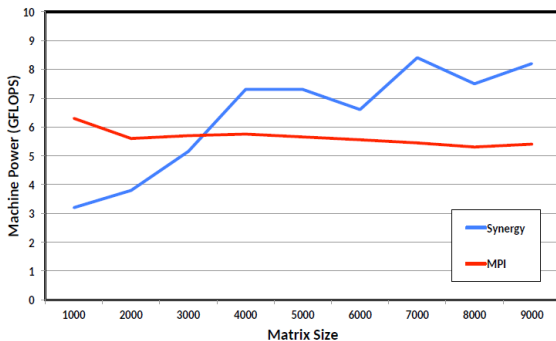


Figure 8. Single-Node Multiple-Core (Broader Range Tests)

7.3. Multiple-Node Multiple-Core Results

multi-node multi-core environment. The Brachistochrone effects are shown again by the multiple optimal synchronization times at specific granularity points. These points are modulated by the increasing communication overheads (left to the optimal granularity (70) and increasing synchronization (waiting) time (right to the optimal point 70).

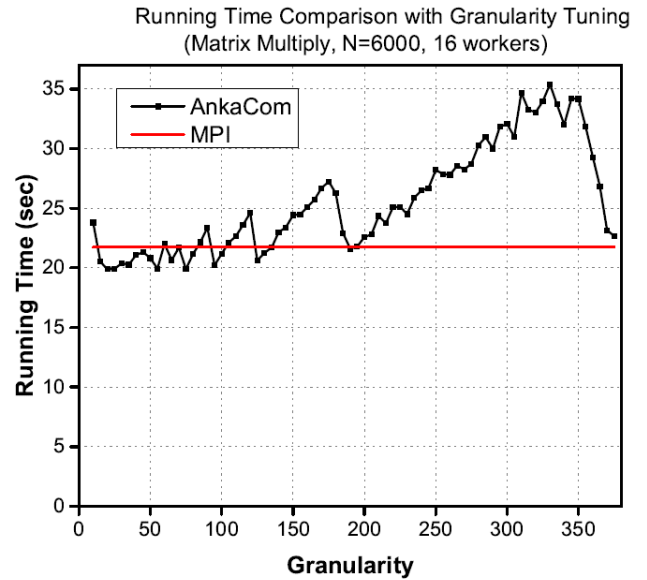


Figure 9. Multi-Node Multi-Core Granularity Tuned

Figure 10 reports AnakaCom performances against MPI using 120 cores.

AnkaCom implements multi-threaded distributed Tuple Space daemons following the UVR architecture. As discussed, this effectively decouples application programs and data completely from processors and networks. Each distributed Tuple Space server can replicate its contents elsewhere ($R > 0$). In this reported experiment, $R = 1$. This means that each tuple is replicated with 1 copy elsewhere. This experiment was designed to reveal the effects of "diminishing of return" as well as the power of granularity tuning.

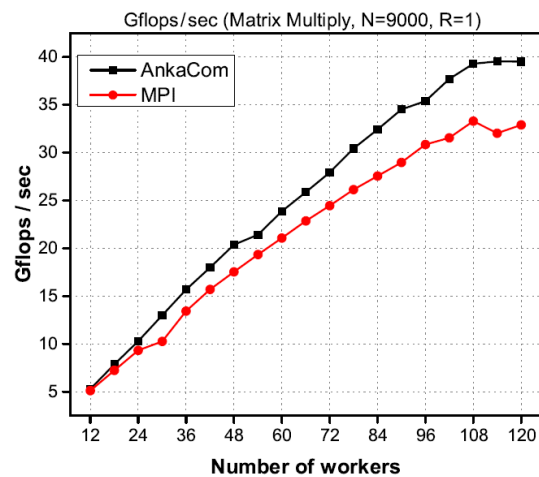


Figure 10. Multi-Node Multi-Core Tests

the granularity tuning effects in the

Each AnkaCom run is optimized by picking the best performing granularity through multiple runs. The recorded MPI run is the best of three runs with identical task size N/P .

Figure 10 shows consistently better AnkaCom performances over MPI. The performance differences grew bigger as the number of cores increased. The MPI performance started to trend down at $P=108$ while AnkaCom kept strong for yet another 12 more cores until the end where N/P became merely 75 (9000/120). Note that the Intel XEON clock cycle is 2.2GHZ. The 75 rows (9000 columns) of dot-products seems the break-even point for sustained performance of this particular application. After this point, both MPI and AnkaCom's communication overheads took over. This was a demonstration of the economic law of "diminishing return".

The C-MPI runs are via the "mpirun -np X -mca btl ib" option triggering the use of custom Infiniband driver. The AnkaCom Infiniband support is by the standard Java library.

8. Summary

Solving the scalability dilemma is of primary importance for developing robust computing services. End-to-end computing paradigms are suitable for small scale data processing needs. In the era of big data, with the Moore's Law approaching its end, extreme scale clustered computing is the only possible path for the future internet-scaled computing. The prior large scale HPC experiments uncovered the scalability dilemma, energy efficiency and reproducibility challenges in delivering robust computing services. It is time to eliminate the known fallacy in the making of distributed computing services.

According to the impossibility theory [2] and the possibility of delivering reliable service using faulty components [4], addressing the scalability challenges requires a paradigm shift: from IP-addressable programming to content addressable programming. This paper reports a single-sided Statistic Multiplexed Computing (SMC) paradigm in order to circumvent the impossibility and to fully leverage the possibility of statistic multiplexing. A narrow Scalable Computing Service definition and three reproducible scalable service tests are proposed. MPI, Hadoop, Spark, Synergy and AnkaCom are all examined using the same scalability tests.

In order to gain a sense of deliverable performance and reliability of the proposed service architecture in multi-node and multi-core environments, computational experiments are conducted in three groups: single node and single core, single node multiple core and multiple node and multiple core. The single-node single-core group provides the baseline for performance comparisons.

The computational results revealed the following:

- Low communication overhead is *not* a necessary condition for the optimized parallel performance. Figures 2-9 demonstrated overwhelmingly that parallel performance optimization by tuning the processing granularity can easily out-perform the low overhead MPI implementation.
- The choice of fixed N/P partition for MPI was deliberate. It is possible to build a task-pool layer in MPI or use single-sided "scatter_v" to allow dynamic granularity tuning without reprogramming. There will be significant implementation challenges considering passing the *Share-nothing* and *Sublinear Cost Tests*.

However, when the problem size, the number of processors and the dynamic runtime load distribution happen to hit the "sweat-spot" (the fixed granularity is the optimal for the given runtime dynamics), MPI program will out-perform Synergy and AnkaCom.

- Both Synergy and AnkaCom were running with built-in fault tolerance. Including checkpoints in the MPI program will push the performance to be much worse than reported.

The application reliability of the single-sided paradigm was also tested. Connection failures did occur during Synergy 48 core tests. The prototype SMC protocols worked flawlessly allowing all tests to complete without any checkpointing.

The single-sided SMC application demonstrated consistently better performance in all experiments. Since SMC applications are natively protected by the TSN, the SMC application's performance and reliability have far exceeded the end-to-end computing MPI programs.

The SMC principle is also applicable for transactional and storage services [5]. Solving the data intensive SCS problem requires a solution to the CAP Theorem [29].

This paper tries to convince the reader that the robust service scalability dilemma of distributed and parallel computing is indeed solvable. The proposed single-sided SMC paradigm is a feasible solution for delivering efficient and robust computing services using faulty components. Correct implementation of the Statistic Multiplexed Computing principle ensures the reliability of large scale distributed services. Extracting the optimal processing performance is similar to finding the Brachistochrone curve in physics and mathematics. The SMC goal of building non-stoppable services is similar to the Etherem project [25]. The implementation of SMC paradigm, however, can promise high information security without linearly increasing overheads.

In addition to extreme scale parallel computing, the single-sided SMC service architecture is also suited for mission critical applications. It promises true non-stop computing service as long as the infrastructure affords the minimal survivable resource set, a task that can be automated completely in practice. This development will impact all existing mission critical distributed and parallel services. These include Enterprise Service Buses (ESB), mission critical transaction processes, mission critical storage systems, smart grid controllers and software defined network (SDN) control plane implementations. Finally, the use of service content addressable network for mission critical services makes all IP-based cyberattacks, such as DDOS (Distributed Denial of Service), ineffective. These developments will redefine the state of the Internet.

About Authors

The corresponding author is responsible for the scalability definition and tests, the UVR architecture design and the first prototype implementation Synergy. He is the original proposer of the Stateless Parallel Processing concept and the SMC framework. Author Yasin Celik is currently a PhD student completing a dissertation at Temple University. Yasin is responsible for the practical rendition of the SMC framework (AnkaCom) and validation tests. His dissertation includes both compute and data intensive SMC implementations and scalability studies.

Acknowledgment

The reported study is supported in part by the 2016 NSF REU Program at Temple University Site, NSF MRI Grant #CNS0958854 (owlsnest.hpc.temple.edu), and computing resource grant #CH-817746 from the NSF Chameleon Computing Cloud Testbed.

The authors thank two NSF REU (Research Experience for Undergraduate) program students, Kimberly Kosman (Purdue University) and Traves Evans (Louisiana Polytechnique Institute) contributed in independent validation of prior performance results in the summer of 2016.

Support from the administrative staff of the Owlsnest.hpc.temple.edu and the CIS Department, College of Science and Technology are also acknowledged.

References

- [1] Peter Deutsch. "Eight fallacies of distributed computing". [online] <https://blogs.oracle.com/jag/resource/Fallacies.html>.
- [2] Alan Fekete, Nancy Lynch, Yishay Mansour, and John Spinelli. "The impossibility of implementing reliable communication in the face of crashes". *J. ACM*, 40:1087–1107, November 1993.
- [3] XSEDE 2014 reproducibility workshop report, "Standing together for reproducibility in large-scale computing". [online] <https://www.xsede.org/documents/659353/d90df1cb-62b5-47c7-9936-2de11113a40f>.
- [4] John M. Spinelli. "Reliable data communication in faulty computer networks", June 1989. PhD thesis, MIT.
- [5] OSI Reference Model [online] https://en.wikipedia.org/wiki/OSI_model
- [6] Justin Y. Shi, Yasin Celik, "AnkaStore: A Single-Sided Statistic Multiplexed Transactional Distributed Storage System", unpublished manuscript, 2017.
- [7] R. S. Nikhil Arvind. "Implicit parallel programming in pH", In Introduction to Management Science (10th Ed), Morgan Kaufmann, 2011.
- [8] David Gelernter and Nicholas Carriero. "Coordination languages and their significance". In *Communications of ACM*. ACM, 1992.
- [9] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. "A scalable content-addressable network". In *SIGCOMM 2001*, San Diego, CA, USA, 2001. ACM.
- [10] Yuan Shi. "Stateless Parallel Processing Prototype: Synergy", 1996. [online] <https://github.com/jys673/Synergy30>.
- [11] Justin Y. Shi. "System for high-level virtual computer with heterogeneous operating systems". *U.S. Patent #5,381,534*, 1995.
- [12] Justin Y. Shi. "Multi-computer system and method". *U.S. Patent #5,517,656*, 1996.
- [13] Justin Y. Shi and Suntian Song. "Apparatus and method of optimizing database clustering with zero transaction loss". *U.S. Patent Application: #2008018969*, 2008.
- [14] Justin Y. Shi. "Fault tolerant self-optimizing multiprocessor system and method thereof". *U.S. Patent Application #20090019258*, 2007.
- [15] Justin Y. Shi. "System and method for fault tolerant scalable computing". *U.S. Patent Application #605725*, November 2016.
- [16] Justin Y. Shi, Moussa Taifi, Abdallah Khreishah, and Jie Wu. "Tuple switching network – when slower maybe better". *International Journal of Parallel and Distributed Computing*, 72, 2011.
- [17] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. "Chord: A scalable peer-to-peer lookup service for Internet applications". *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.
- [18] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the worldwide web". In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.
- [19] Y. Shi. "A distributed programming model and its applications to computation intensive problems for heterogeneous environments". In *1992 Earth and Space Science Information Systems, AIP Conference Proceedings*, Pasadena, CA, 1992.
- [20] Justin Y. Shi, Moussa Taifi, Abdallah Khreishah, Aakash Pradeep, and Vivek Anthony. "Program scalability analysis for HPC cloud: Applying Amdahl's law to NAS benchmarks". In *International Workshop on Sustainable HPC Cloud/Supercomputing Conference 2012*. IEEE, 2012. Salt Lake City, Utah.
- [21] Brachistoschrone curve. [online] <https://en.wikipedia.org/wiki/Brachistochrone>.
- [22] Judith Hippold and Gudula Runger. "Task pool teams for implementing irregular algorithms on clusters of SMPs". Nice, France, 2003. IEEE Press.
- [23] Apache hadoop. <http://hadoop.apache.org>.
- [24] Archley Kattt. "Spark – lightning-fast cluster computing". Berkeley, CA, USA, 2011. University of California.
- [25] Ethereum Homestead Documentation – "A Blockchain Application Platform" [online] <http://www.ethdocs.org/en/latest/>.
- [26] Yasin Celik, Aakash Pradeep, and Justin Y. Shi. Ankacom: "A development and experiment for extreme scale computing". In *CIT/IUCC/DASC/PICom*, pages 2010–2016. IEEE, 2015.
- [27] Chameleon. "A configurable experimental environment for large-scale cloud research". <https://www.chameleoncloud.org/>, 2017.
- [28] Temple University HPC Facility: Owlsnest. <http://www.hpc.temple.edu/owlsnest/OwlsnestUserGuide.html>.
- [29] Gilbert and Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". In *ACM SIGACT News (2002)*, volume 33(2), page 59. ACM, 2002.