PhD Proposal – Dynamic Distributed Execution Using Declarative Programming

Karthik Nilakant

1 Overview

A focus area of current systems research is on parallelism within a computer and across computers. Computation can be manually or automatically distributed over large clusters of commodity hardware, for instance within a datacentre. Parallel processing on a massive scale has allowed organisations to quickly process vast quantities of data, or to provide this capability as a service to others.

Google's large scale distributed execution framework, MapReduce [4], was the foundation for a number of projects that have extended and enriched the functionality. MapReduce had been used for several years as Google's primary means of indexing its crawled content store, as well as for other large batch processes such as image processing for Google Earth. Although a variety of distributed algorithms can be expressed using the map and reduce primitives, some problems call for a more complex arrangement of operations, for example multiple map and reduce steps, or operations on two different batches of data that are reduced into a single output set. The arrangement of operations can be defined in terms of a directed acyclic graph, known as a "task graph". A distributed execution engine can then assign each task, or parallel instances of the same task, to nodes in the cluster. This is the approach taken by Dryad [10], which can be used to express MapReduce-style operations, as well as a range of other task graphs. However, the task graphs must be defined statically prior to program execution. Static task graphs cannot be used to express programs where the control flow is dependent on the data, such as loops that that are conditional on a convergence test. To allow this type of scenario, the execution engine must be able to modify the task graph during execution. This is one of the main features of the Ciel execution engine, which can be used to express programs of arbitrary complexity [14].

The task graphs that are used by Ciel, Dryad and MapReduce are defined programmatically. In each case, a non-imperative programming language has been used - in the case of Ciel, Skywriting is a functional language based on JavaScript, whereas DryadLINQ uses the language integrated query specification for Microsoft's .NET language family. Declarative languages allow developers to define the solution in terms of the generic task flow, without having to concern themselves with the underlying distribution of tasks among nodes. Similarly, data flows connected by operators have been used as an abstraction in the field of data stream processing. In these applications, data continuously flows from sources such as sensors, financial indicators or other rapidly changing sources through a series of operators that may aggregate or transform the data from each stream, to an eventual data sink. Some research has aimed to introduce further parallelism into these stream processing flows [8].

Clearly, there is an opportunity to take advantage of the similarities in these various approaches to data-centric programming. A system that facilitates the convergence of these approaches may have the following properties:

- The system would be driven by a declarative programming language, that is expressive enough to handle problems involving both batches of data and continuous streams, and allow control flow constructs so that Turing-completeness can be achieved.
- The system would utilise a distributed execution engine that is able to synthesise task graphs from the aforementioned language at runtime, and decide on an optimal execution strategy using any available processors at any given instant. It should have the ability to distribute execution across a range of heterogeneous computational modules. This could include multi-core, many-core, cloud and mobile ad-hoc computing models.
- The environment could facilitate execution of operations on processors with varying levels of associated trust. For instance, machines within the same data centre as the data source may be physically

secured and therefore trusted, whereas utility computing capacity obtained through a cloud provider may be completely untrusted.

The following sections will elaborate on these properties.

2 Adaptable Declarative Programming

The use of declarative and functional programming languages for parallel processing is a current area of research. The main impetus for this is that these languages allow processes to be defined without any so-called "side effects", meaning execution can easily be divided and distributed to other machines. Indeed, a branch of declarative languages (known as "dataflow" languages) have been widely adopted in the development of firmware for programmable logic controllers and FPGAs. Languages such as Verilog and VHDL allow the developer to define blocks of sequential logic that are executed concurrently on hardware.

Declarative languages focus on what the result of a computation should be, rather than on how the result should be computed. As a result, programs written with a declarative language are usually shorter than those written with an imperative language, and therefore are (arguably) easier to debug and maintain. It is also typically less complicated to verify a declarative program for correctness. On the other hand, the control afforded by an imperative language can be essential for tuning an algorithm for optimal performance. Typically, declarative programming languages tend to impose a significant performance penalty, however in recent times the gap may have narrowed somewhat. In the BOOM Analytics project, declarative programming has been used to reduce the code complexity of an existing distributed infrastructure [1].

A technique used in some languages that have been built for distributed execution is to allow the developer to "escape" into an imperative syntax for portions of code as required. This has been used in DryadLINQ, in the form of the Apply keyword, and also within the Flask functional language, where developers can define blocks of NesC code, to optimise programs for sensor network motes [13]. However, these blocks may impede the ability of a semantic checker to verify a program, due to the presence of imperative code fragments.

Another approach which allows the declarative syntax to be preserved is query optimisation. The declarative language SQL has been used for data definition and manipulation in relational databases for many years. Modern RDBMS query optimisers construct task graphs (known as query plans) that can be used to resolve a SQL query. In cases where multiple possible query plans exist, the query optimiser will choose the least-cost query using heuristics and statistics that have been collected about the database. However, for this to work efficiently, the database administrator needs to ensure accurate statistics are being recorded, and to add indexes or change the physical table layout to improve performance – essentially, the data is optimised in order to optimise the program [5].

Since it is desirable for programs written in this language to run on multiple heterogeneous devices (as explained in the following section), the language also needs to be suitably abstract so that it does not constrain the type of devices that can be supported.

3 Distributed Heterogeneous Computation

In current parlance, heterogeneous computing is used to refer to parallel programs that execute on processors with varying instruction set architectures (ISAs), but within the same machine (that is, with access to the same memory bus). Distributed computing is, on the other hand, used to refer to an application that runs concurrently on a set of independent machines, connected by a computer network. A framework that combines these two models might allow programs to execute concurrently on various processor cores within a machine, as well as on various cores on external machines, each with varying architectures and performance characteristics. Any distributed system is, in a sense, also heterogeneous, however it is rare to see a program that can utilise the computing resources of mobile phones and servers in the cloud in a unified fashion.

Heterogeneity brings about a number of technical challenges for a distributed execution framework. Firstly, a means to execute code on each type of device is required, be it in the form of a virtual machine / runtime environment, an interpreter or a just-in-time compiler to allow the code to run natively. Each of these approaches has various costs and benefits in terms of performance and flexibility. Furthermore, some approaches may simply not be possible in certain situations.

Secondly, non-deterministic factors can have varying effects even on systems which would otherwise be regarded as homogeneous, which produces another form of heterogeneity. For examples, a server may be running some other workload, a network connection may be congested, disk storage may be highly fragmented, or physical parts may become degraded. In order to ensure optimum efficiency, the execution engine needs to track the performance of these various components and redistribute the task graph appropriately.

Flask is an example of an existing system that allows operators in these systems to be defined using a functional language, and the code is compiled down to native binaries that can be installed on individual sensor nodes. Another system monitors the latency and bandwidth of network links that connect a set of operators, and dynamically adjusts the location of those operators to optimise the efficiency of the process.

4 Secure Opportunistic Communications

Most existing large-scale parallel execution systems, for instance BOINC and Condor which are used as part of the LHC Computing Grid, rely on a centralised scheduling server [12, 2]. Existing distributed execution engines such as MapReduce also use a similar single-master scheme. Although the master itself has very little work to do in the actual computation of each task, it can be heavily loaded by the volume of tasks in a large-scale parallel computation. It would also be desirable to avoid the single point of failure that a single master entails. Some proposed solutions to this have included the use of "standby masters", which take over the role of the master when it fails. Google's infrastructure uses a distributed locking system called Chubby to achieve this [3].

Another approach would be to allow the client that submits a job to act as the master. A node could utilise any available computational resources that are "local" to the node in an opportunistic manner. "Locality" in this environment can have various semantics. In terms of performance, the initiating node may seek to distribute tasks to nodes with the lowest network latency and highest throughput available - for example, cores within the local host would be preferred versus cores on the local network. The typical tradeoff is that a greater number of processors are available at a greater "distance" from the node, meaning the execution engine will need to balance the degree of parallelism against the communication costs. The SBON project uses a flexible cost optimisation scheme to achieve this [15].

Locality can also be extended to the concept of trust among nodes. The single master model allows the scheduler to act as a "gatekeeper" for the cluster, which means that trust can be managed hierarchically. Hierarchical trust management is also utilised in the public key infrastructure [9]. In many cases, the level of trust in a remote system aligns with the network distance. For instance, in an organisational network, common ownership and control of the devices within the local network increases the level of trust between those devices. To utilise resources outside the network, assumptions of trust may need to be relaxed or foregone completely.

Managing trust in a decentralised manner has been attempted previously, for example with PGP's "web of trust" model [21]. In these systems, groups of of users or machines that explicitly trust each other can endorse each other's digital certificates. However, newly published certificates will take time to build up enough endorsements to be usable on the network. For individuals, certificate management could be integrated with online social networks to speed up this process, allowing new users to be referred to other friends in the network, expediting the endorsement process. However, for commercial organisations, the hierarchical model may be a better trust management approach.

Regardless of the approach, security safeguards will still need to be employed to protect against malicious actions. Data-centric networking models allows data to flow from node to node without coupling to specific endpoints [11, 6, 20]. This is especially useful where multiple nodes may need access to the data, to allow for parallel processing and fault tolerance. It also allows the content itself to be signed and encrypted, rather than requiring endpoints to communicate over an encrypted channel. This means that untrusted nodes could potentially be used to temporarily store data in encrypted form. Advances in homomorphic encryption could possibly even allow untrusted nodes to execute encrypted code on encrypted data, without needing to decrypt it beforehand [7, 17]. An opportunistic execution engine could decide, depending on what is known about the layout of the network, whether a given program could be safely executed on untrusted nodes.

Security must also be provided against the converse issue: untrusted code must not be allowed to compromise a computing resource, either intentionally or otherwise. This is a well-researched area of computer security, and solutions for this include application / operating system isolation, processor privilege denial, and resource partitioning. Grid systems such as BOINC make use of idle processor cycles by modifying the thread priority of external tasks on the system scheduler.

5 Research Goals and Provisional Plan

To summarise the previous sections, we desire to execute programs written in a declarative language, making optimal use of the parallel processing power provided by a set of heterogeneous devices connected by an opportunistic network, with varying levels of trust between each device.

The aim of this project will be to identify a range of possible options for the components of such a system, and determine the costs and benefits of each alternative by empirical evaluation. This will involve building an extensible framework for each of the core functions of the system: a declarative language syntax and associated compiler or interpreter, a dynamic distributed execution engine, and a secure opportunistic communications platform.

5.1 Plan for First Year

- Existing declarative languages that have been used in similar research, such as SkyWriting and DryadLINQ [19], typically take a functional approach. This is seen as a middle ground between imperative programs and purely declarative programs. However, a need for flexibility in either direction exists. In particular, a desirable attribute would be to increase the level of abstraction in the language, while allowing annotations to "hint" at possible execution plans for the compiler / interpreter. These annotations may also define a "level of trust" required for specific code fragments. To this end, further study of existing languages that take this approach is needed, in order to synthesize a syntax for an initial declarative language that can be used with this system. (Estimate 1-3 months of continuous effort. Deliverables: survey report of existing work on parallel declarative languages, proposed syntax of new language).

- A range of existing projects provide functionality for components and sub-components of the framework, as described previously [16, 14, 18]. An initial task will be to survey existing work, and identify any gaps in functionality. It will also be useful to build up a baseline system consisting of existing components, that could be used as a performance benchmark for further work. This also means that a means for trialling and measuring the performance of the system would need to be devised. (Time estimate 2-4 months. Deliverables: functional gap analysis, empirical results based on trial of existing components).

- Existing load distribution approaches focus on identifying "stragglers" by measuring the rate of progress, or estimating the end time of each executing task. However, these cost calculations typically do not take into account the network-related costs of transferring data to/from each processing node. To illustrate this concept, an existing execution engine will be adapted to use a modified load distribution scheme, which incorporates a combination of progress tracking and network latency / bandwidth considerations, to allow operation in heterogeneous "multi-cloud" environments. (Time estimate 2-4 months. Deliverables: modular extension to existing system, technical report of optimisation approach and results of trial).
- Advances in homomorphic encryption could make it usable for practical applications. However, the space and time complexity of current approaches are still high-degree polynomial functions of the desired level of security. It may be possible to improve the performance of current techniques by introducing parallelism in the various algorithms involved. A prototype will be built to assess the feasibility of this approach. If the results are promising, it will likely define the shape of the security architecture for the emerging system. (Estimate 3-6 months of continuous effort. Deliverable: technical report / paper describing implementation on distributed infrastructure and real-world trial).

5.2 Subsequent Effort

A compiler and interpreter will be constructed for the declarative language defined previously. By comparing and contrasting the two approaches, it should be possible to ascertain if there is a clear advantage to exclusively using one approach over the other, or if a hybrid approach is warranted. To verify that the declarative approach is indeed capable of improving the quality of code, a basic code verification system will be investigated. In addition, options for distributed tracing and debugging will be explored.

Further extensions to the execution engine will be explored, by allowing the system to be extended to utilise different types of devices or hardware components. This could include multiple cores in a system such as GPUs, mobile devices, sensor motes, or pre-configured groupings of devices such as globally distributed desktop clusters. The aim will be capture a broad spectrum of heterogeneity, and show how the system can dynamically adapt to take advantage of the potential processing power of an environment involving any or all of these computers.

Finally, the security and communications framework will be extended by considering options for key management and federated trust, in both a centralised and decentralised manner. The final thesis will focus on how tight integration between these three streams can allow concepts that were once regarded as liabilities, such as heterogeneity, distrust and abstraction can become essential assets in a unified parallel execution system.

References

[1] ALVARO, P., CONDIE, T., CONWAY, N., ELMELEEGY, K., HELLERSTEIN, J. M., AND SEARS, R. Boom analytics: exploring data-centric, declarative programming for the cloud. In *Proceedings of the 5th European conference on Computer systems* (New York, NY, USA, 2010), EuroSys '10, ACM, pp. 223–236.

[2] ANDERSON, D. P. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing* (Washington, DC, USA, 2004), GRID '04, IEEE Computer Society, pp. 4–10.

[3] BURROWS, M. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation* (Berkeley, CA, USA, 2006), OSDI '06, USENIX Association, pp. 335–350.

[4] DEAN, J., AND GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM 51* (January 2008), 107–113.

[5] ELMASRI, R., AND NAVATHE, S. *Fundamentals of Database Systems*, 6th ed. Addison-Wesley Publishing Company, USA, 2010.

[6] ESTEVE, C., VERDI, F. L., AND MAGALHÃES, M. F. Towards a new generation of information-oriented internetworking architectures. In *Proceedings of the 2008 ACM CoNEXT Conference* (New York, NY, USA, 2008), CoNEXT '08, ACM, pp. 65:1–65:6.

[7] GENTRY, C. Computing arbitrary functions of encrypted data. *Commun. ACM 53* (March 2010), 97–105.

[8] GULISANO, V., JIMENEZ-PERIS, R., PATINO-MARTINEZ, M., AND VALDURIEZ, P. Streamcloud: A large scale data streaming system. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems* (Washington, DC, USA, 2010), ICDCS '10, IEEE Computer Society, pp. 126–137.

[9] HOUSLEY, R., POLK, W., FORD, W., AND SOLO, D. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, 2002.

[10] ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (New York, NY, USA, 2007), EuroSys '07, ACM, pp. 59–72.

[11] JOKELA, P., ZAHEMSZKY, A., ESTEVE ROTHENBERG, C., ARIANFAR, S., AND NIKANDER, P. Lipsin: line speed publish/subscribe inter-networking. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication* (New York, NY, USA, 2009), SIGCOMM '09, ACM, pp. 195–206.

[12] LAMANNA, M. The lhc computing grid project at cern. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 534*, 1-2 (2004), 1 – 6. Proceedings of the IXth International Workshop on Advanced Computing and Analysis Techniques in Physics Research.

[13] MAINLAND, G., MORRISETT, G., WELSH, M., AND NEWTON, R. Sensor network programming with flask. In *Proceedings of the 5th international conference on Embedded networked sensor systems* (New York, NY, USA, 2007), SenSys '07, ACM, pp. 385–386.

[14] MURRAY, D. G., AND HAND, S. Scripting the cloud with skywriting. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 12–12.

[15] PIETZUCH, P., LEDLIE, J., SHNEIDMAN, J., ROUSSOPOULOS, M., WELSH, M., AND SELTZER, M. Network-aware operator placement for stream-processing systems. In *Proceedings of the 22nd International Conference on Data Engineering* (Washington, DC, USA, 2006), ICDE '06, IEEE Computer Society, pp. 49–.

[16] SCOTT, J., CROWCROFT, J., HUI, P., AND DIOT, C. Haggle: a networking architecture designed around mobile users.

[17] SMART, N., AND VERCAUTEREN, F. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography PKC 2010*, P. Nguyen and D. Pointcheval, Eds., vol. 6056 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2010, pp. 420–443.

[18] YONEKI, E., BALTOPOULOS, I., AND CROWCROFT, J. D3n: programming distributed computationin pocket switched networks. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds* (New York, NY, USA, 2009), MobiHeld '09, ACM, pp. 43–48.

[19] YU, Y., ISARD, M., FETTERLY, D., BUDIU, M., ERLINGSSON, U., GUNDA, P. K., AND CURREY, J. Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation* (Berkeley, CA, USA, 2008), OSDI'08, USENIX Association, pp. 1–14.

[20] ZHANG, L., ET AL. Named Data Networking (NDN) Project, Oct. 2010.

[21] ZIMMERMANN, P. R. *The official PGP user's guide.* MIT Press, Cambridge, MA, USA, 1995.