WEBCOM: A WEB BASED DISTRIBUTED COMPUTATION PLATFORM

JOHN P. MORRISON, DAVID A. POWER, JAMES J. KENNEDY
Centre for Unified Computing,
Department of Computer Science,
National University of Ireland,
Cork, Ireland.
http://www.cuc.ucc.ie

Abstract
The propagation of the World Wide Web into everyday use, coupled with Java technologies, provide a foundation upon which a distributed computing environment can easily be constructed. The WebCom system presented in this paper represents an attempt at implementing one such environment. The system operates by distributing 'instructions' to client machines for execution. These instructions are created using an inherently parallel model of computation capable of utilising data-driven, demand-driven and imperative computations.

Keywords: *Parallel, Distributed, Metacomputing, Dataflow, Condensed Graphs, Java.*

1. INTRODUCTION

Existing web based distributed computing systems[2, 4, 8, 9] involve the distribution of Java objects to client machines which are connected using applets embedded within web pages or specialised Java applications. Java applications are used so that the well known features of applet security can be avoided, providing greater access to client machines. Application developers are required to determine the parallelism available in a particular problem and to use libraries provided in conjunction with Remote Method Invocation (RMI) and Object Request Brokering (ORB) to create a distributed system to suit the problem. WebCom uses similar ideas, however, a much more simplistic approach to communication is adopted: only one generic applet is employed and both RMI and ORB are avoided. WebCom uses a graph based model of computation[10, 11] to generate the instructions that will be executed on client machines. This model is similar to the dataflow model [1, 5, 7, 13] but can also exploit demand-driven and imperative computations simultaneously. This model implicitly exposes parallelism, however both the programmer and feedback from underlying machine characteristics can influence the execution profile. The Condensed Graphs model is hierarchical. Nodes are used to represent basic instructions or other Condensed Graphs. An extended version of the WebCom architecture allows for the possibility for clients to act as masters. In this way, a complete Condensed Graph may be

sent to a client for execution. In the promotion process, other clients are reassigned to execute work generated by the new master.

2. CONDENSED GRAPHS

While being conceptually as simple as classical dataflow schemes [7, 3], the Condensed Graphs ($\mathcal{CG}$) model is far more general and powerful. It can be described concisely, although not completely, by comparison. Classical dataflow is based on data dependency graphs in which nodes represent operators and edges are data paths which convey simple data values between them. Data arrive at *operand ports* of nodes along input edges and so trigger the execution of the associated operator (in dataflow parlance, they cause the node to *fire*). During execution, these data are consumed and a resultant datum is produced on the node's outgoing edges. This result acts as input to successor nodes. Operand sets are used as the basis of the firing rules in data-driven systems. These rules may be *strict* or *non-strict*. A strict firing rule requires a complete operand set to exist before a node can fire; a non-strict firing rule triggers execution as soon as a specific proper subset of the operand set is formed. The latter rule gives rise to more parallelism but also can result in overhead due to remaining packet garbage (RPG).

Like classical dataflow, the $\mathcal{CG}$ model is graph-based and uses the flow of entities on arcs to trigger execution. In contrast, $\mathcal{CG}$s are directed acyclic graphs in which every node contains not only operand ports, but also an operator and a destination port. Arcs incident on these respective ports carry other $\mathcal{CG}$s representing operands, operators and destinations. Condensed Graphs are so called because their nodes may be condensations, or abstractions, of other $\mathcal{CG}$s. (Condensation is a concept used by graph theoreticians for exposing meta-level information from a graph by partitioning its vertex set, defining each subset of the partition to be a node in the condensation, and by connecting those nodes according to a well-defined rule [6].) Condensed Graphs can thus be represented by a single node (called a *condensed node*) in a graph at a higher level of abstraction. The number of possible abstraction levels derivable from a specific graph depends on the number of nodes in that graph and the partitions chosen for each condensation. Each graph in this sequence of condensations represents the same information but in a different level of abstraction. It is possible to navigate between these abstraction levels, moving from the specific to the abstract through condensation, and from the abstract to the specific through a complementary process called *evaporation*.

The basis of the $\mathcal{CG}$ firing rule is the presence of a $\mathcal{CG}$ in every port of a node. That is, a $\mathcal{CG}$ representing an operand is associated with every operand port, an operator $\mathcal{CG}$ with the operator port and a destination $\mathcal{CG}$ with the destination port. This way, the three essential ingredients of an instruction are brought together (these ingredients are also present in the dataflow model; only there, the operator and destination are statically part of the graph).

A condensed node, a node representing a datum, and a multi-node $\mathcal{CG}$ can all be operands. A node represents a datum with the value on the *operator* port of the node. Data are then considered as zero-arity operators. Datum nodes represent graphs which cannot be evaluated further and so

are said to be in *normal form*. Condensed node operands represent unevaluated expressions. They cannot be fired since they lack a destination. Similarly, multi-node $\mathcal{CG}$ operands represent partially evaluated expressions. The processing of condensed node and multi-node operands is discussed below.

Any $\mathcal{CG}$ may represent an operator. It may be a condensed node, a node whose operator port is associated with a machine primitive (or a sequence of machine primitives) or it may be a multi-node $\mathcal{CG}$.

The present representation of a destination in the $\mathcal{CG}$ model is as a node whose own destination port is associated with one or more port identifications. The expressiveness of the $\mathcal{CG}$ model can be increased by allowing any $\mathcal{CG}$ to be a destination but this is not considered further here. Fig. 1 illustrates the congregation of instruction elements at a node and the resultant rewriting that takes place.

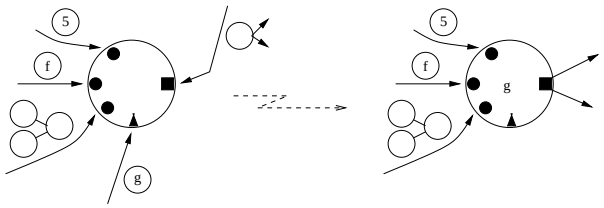

Figure 1: $\mathcal{CG}$s congregating at a node to form an instruction

When a $\mathcal{CG}$ is associated with every port of a node it can be fired. Even though the $\mathcal{CG}$ firing rule takes account of the presence of operands, operators and destinations, it is conceptually as simple as the dataflow rule. Requiring that the node contain a $\mathcal{CG}$ in every port before firing prevents the production of RPG. As outlined below, this does not preclude the use of non-strict operators or limit parallelism.

A *grafting* process is employed to ensure that operands are in the appropriate form for the operator: non-strict operators will readily accept condensed or multi-node $\mathcal{CG}$s as input to their non-strict operands. Strict operators require all operands to be data. Operator strictness can be used to determine the strictness of operand ports: a strict port must contain a datum $\mathcal{CG}$ before execution can proceed, a non-strict port may contain any $\mathcal{CG}$. If, by computation, a condensed or multi-node $\mathcal{CG}$ attempts to flow to a strict operand port, the *grafting* process intervenes to construct a destination $\mathcal{CG}$ representing that strict port and sends it to the operand.

The grafting process thus facilitates the evaluation of the operand by supplying it with a destination and, in a well constructed graph, the subsequent evaluation of that operand will result in the production of a $\mathcal{CG}$ in the appropriate form for the operator. The grafting process, in conjunction with port strictness, ensures that operands are only evaluated when needed. An inverse process called *stemming* removes destinations from a node to prevent it from firing.

Strict operands are consumed in an instruction execution but non-strict operands may be either consumed or propagated. The $\mathcal{CG}$ operators can be divided into two categories: those that are "value-transforming" and those that only move $\mathcal{CG}$s from one node to another in a well-defined manner. Value-transforming operators are intimately connected with the underlying machine and can range from simple arithmetic operations to the invocation of sequential subroutines and

may even include specialized operations like matrix multiplication. In contrast, $\mathcal{CG}$ moving instructions are few in number and are architecture independent. Two interesting examples are the condensed node operator and the `filter` node. Filter nodes have three operand ports: a Boolean, a `then` and an `else`. Of these, only the Boolean is strict. Depending on the computed value of the Boolean, the node fires to send either the `then` $\mathcal{CG}$ or the `else` $\mathcal{CG}$ to its destination. In the process, the other operand is consumed and disappears from the computation. This action can greatly reduce the amount of work that needs to be performed in a computation if the consumed operands represent an unevaluated or partially evaluated expression. All condensed node operators are non-strict in all operands and fire to propagate all their operands to appropriate destinations in their associated graph. This action may result in condensed node operands (representing unevaluated expressions) being copied to many different parts of the computation. If one of these copies is evaluated by grafting, the graph corresponding to the condensed operand will be invoked to produce a result. This result is held local to the graph and returned in response to the grafting of the other copies. This mechanism is reminiscent of parallel graph reduction [12] but is not restricted to a purely lazy framework.

By statically constructing a $\mathcal{CG}$ to contain operators and destinations, the flow of operand $\mathcal{CG}$s sequences the computation in a dataflow manner. Similarly, constructing a $\mathcal{CG}$ to statically contain operands and operators, the flow of destination $\mathcal{CG}$s will drive the computation in a demand-driven manner. Finally, by constructing $\mathcal{CG}$s to statically contain operands and destinations, the flow of operators will result in a control-driven evaluation. This latter evaluation order, in conjunction with side-effects, is used to implement imperative semantics. The power of the $\mathcal{CG}$ model results from being able to exploit all of these evaluation strategies in the same computation, and dynamically move between them, using a single, uniform, formalism.

3. IMPLEMENTATION

The WebCom system consists of a single master serving an arbitrary number of clients, called Abstract Machines (AMs). The master consists of four parts. A standard web server, an Instruction Constructor (IC), an Instruction Manager (IM) and a Result Handler (RH). The IC makes instructions and passes them to the IM which distributes them to the AMs. The AM is an essential part of the WebCom system. It receives instructions from the IM and executes them to produce results. These instructions can be simple machine operations, sequential programs or complex Condensed Graphs. On execution the AM creates two socket connections to its master, one for receiving instructions and one to send results. When these connections are established the AM reads an instruction, executes it and sends a result back to the RH of the master. The returned results, once processed by the IC, propagate through the computation graph enabling the execution of the destination instructions. This process continues for the duration of the computation.

Socket connections between the AMs and the master are held open continuously, avoiding potential instabilities in the network handling of the JVM (See bug 4032593 in the Java Developer Connection http://developer.java.sun.com). This decision also reduces communications

costs since repeated unsuccessful attempts at opening connections can be expensive. Each of these open sockets are denoted by separate descriptor threads in the master. The IM consults a list of these descriptors to manage all the available AMs. This information can be used for optimisation purposes such as parallelism throttling, error recovery and load balancing.

4. RESULTS

A number of experiments were run to exercise WebCom using the graph depicted in Fig. 2. Each

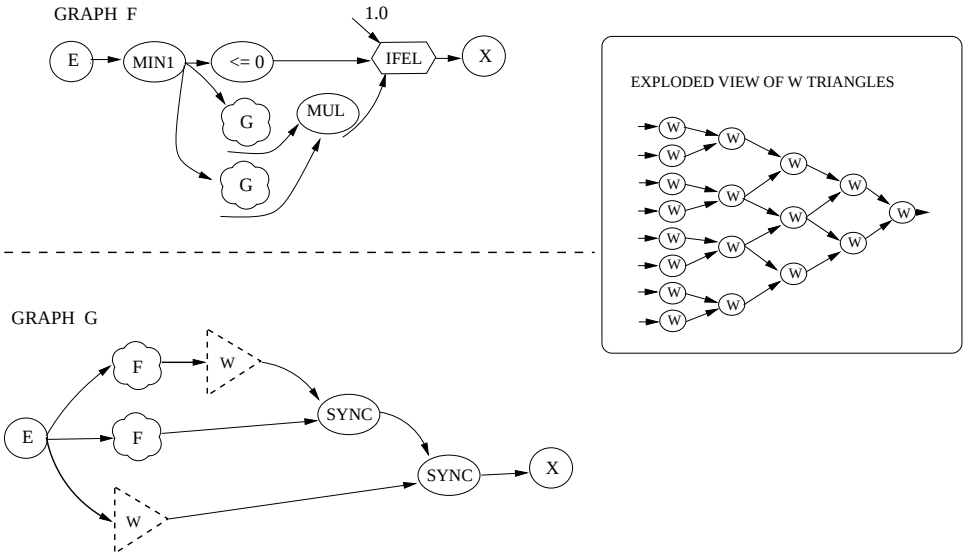

Figure 2: Test execution graph

of the `w` nodes are primitive instructions whose grain-size can be changed from execution to execution. Functions `F` and `G` represent condensed nodes which when executed increase the total number of instructions to be executed. In the experiments each machine is a 266MHz Pentium II with the clients running Linux and the master running Windows NT. The graph was executed using a varying number of AMs and grain-sizes as illustrated in Fig. 3. This figure illustrates that the execution time is reduces as AMs are added to the computation. Moreover, the amount of speedup increases with grain-size. For a given problem size the percentage utilisation of each AM decreases in proportion to the number of AMs available to the master. The rate of this decrease is also proportional to the instruction grain-size. This is illustrated in Fig. 4. As expected, the communication costs does not vary considerably with grain-size but does change with respect to the number of AMs used. In this example the communication overhead falls rapidly from one to four AMs. It then remains relatively constant to eight AMs. Whereas the subsequent addition of AMs results in a decrease in the total execution time, the communication overhead begins to increase proportionately as more time is spent managing communication channels. This is illustrated in Fig. 5.

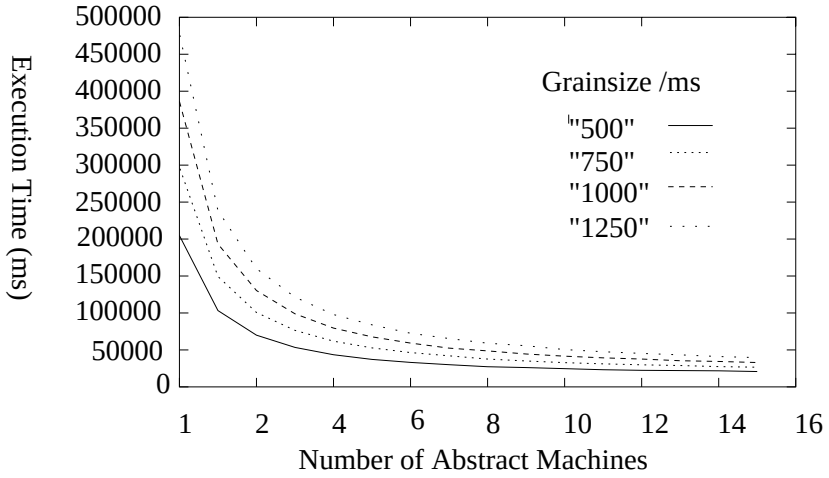

Figure 3: Execution profile for varying Grain-size and AMs

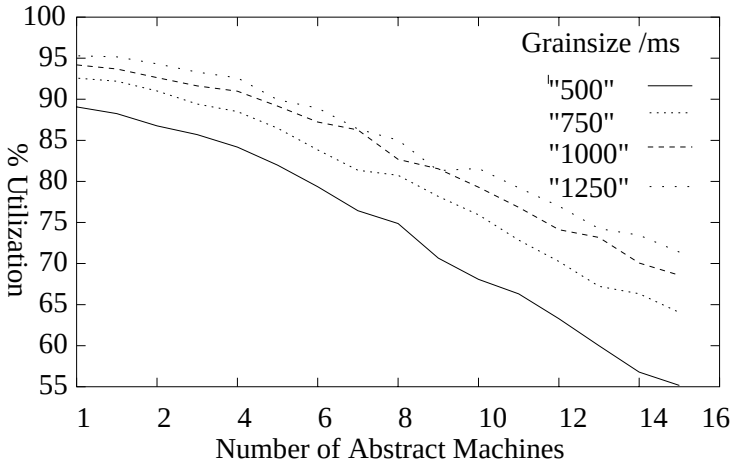

Figure 4: Percentage utilisation of AMs for varying Grain-size

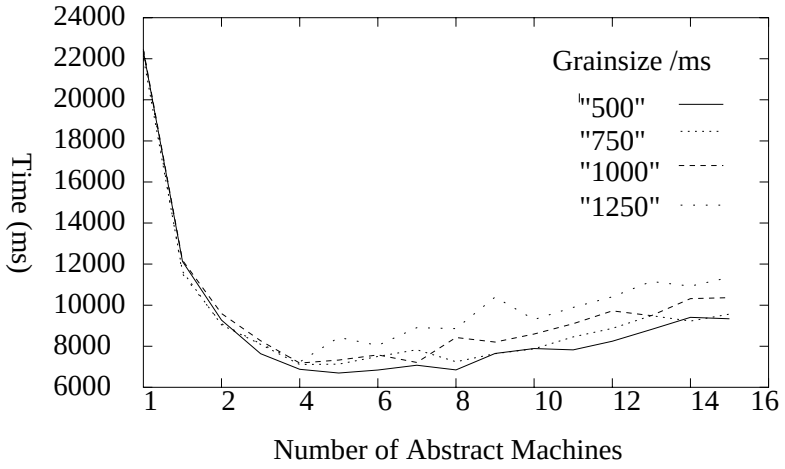

Figure 5: Communication profile for varying Grain-size and AMs

5. IN SUMMARY

WebCom is a Metacomputer for executing Condensed Graphs which exploits common, widely available infrastructures with a minimal installation procedure. It utilises free cycles on client machines to execute programs in a distributed manner. Our results indicate that this is a viable distributed computing platform for instructions of sufficient grain size. In this implementation these instructions can be arbitrarily complex sequential programs or even complex Condensed Graphs which are executed through a client-to-master promotion scheme. Our approach utilises a lightweight communications protocol and recognizes the difficulties inherent in wide area fault tolerant communications. In addition, pleasing characteristics of the Condensed Graphs model, such as variable grain of execution and the combination of data-driven, demand-driven and imperative computations from a single, simple firing rule continue to offer unique solutions to emerging problems. For example, network and processor load balancing is achieved through promoting one or more clients and redirecting others to service the newly promoted masters. This allows a condensed node in a graph to be viewed as a single instruction on one promoted master but executed as a graph on another, diverting processing requirements and network traffic from the original source. The WebCom system illustrates how the Condensed Graphs model of execution can be combined with existing hardware to produce a flexible system of distributed execution.

REFERENCES

[1] D. A. Adams. *A computational model with dataflow sequencing*. PhD thesis, Stanford, California, 1968. TR/CS-117.

[2] Zvi Kedem Arash Baratloo, Mehmet Karul and Peter Wyckoff. Charlotte: Metacomputing on the web. 9th International Conference on Parallel and Distributed Computing Systems, 1996.

[3] Arvind and Kim P. Gostelow. A computer capable of exchanging processors for time. Information Processing 77 Proceedings of IFIP Congress 77 Pages 849-853, Toronto, Canada, August 1977.

[4] P. Cappello, B. O. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-Based Parallel Computing Using Java. In Geoffrey C. Fox and Wei Li, editors, *ACM Workshop on Java for Science and Engineering Computation*, June 1997.

[5] A. L. Davis and R. M. Keller. Dataflow program graphs. In *IEEE Computer Magazine, Vol 15, No 2*, pages 26 – 41, Feb 1982.

[6] Robert Norman Frank Harary and Dorwin Cartwright. Structural models: An introduction to the theory of directed graphs. John Wiley and Sons,1969.

[7] J.R. Gurd, C. C. Kirkham, and I. Watson. The manchester prototype dataflow computer. *Communications of the ACM*, 28(1):34–52, January 1985.

[8] Kwong-Sak Leung, Kin-Hong Lee, and Yuk-Yin Wong. Djm: A global distributed virtual machine on the internet. *Software Practice and Experience*, 28(12), October 1998.

[9] Satoshi Hirano Luis F. G. Sarmenta and Stephen A. Ward. Towards bayanihan: Building an extensible framework for volunteer computing using java. ACM 1998 Workshop on Java for High-Performance Network Computing, Palo Alto, California, Feb. 28 - Mar. 1, 1998.

[10] John P. Morrison. *Condensed Graphs: Unifying Availability-Driven, Coercion-Driven and Control-Driven Computing*. PhD thesis, Eindhoven, 1996.

[11] John P. Morrison and Niall J. Dalton. Condensed graphs: A multi-level, parallel, abstract machine. 13th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'99),Queen's University, Kingston, Canada, 13-16 June 1999.

[12] Rinus Plasmeijer and Marko van Eekelen. Functional programming and parallel graph reduction. ISBN: 0-201-41663-8 Addison-Wesley Publishers Ltd.

[13] Keneth R. Traub, Gregory M. Papadopoulos, Michael J. Beckerle, James E. hicks, and Jonathan Young. Overview of the monsoon project. Technical report, Massachusetts Institute of Technology, January 1991.