RESEARCHERID
THOMSON REUTERS

ISSN: 2277-9655
[Gurusamy * *et al.*, 7(1): January, 2018] Impact Factor: 4.116
IC™ Value: 3.00 CODEN: IJESS7

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

IBIS: THE NEW ERA FOR DISTRIBUTED COMPUTING

Vairaprakash Gurusamy[*1] & K.Nandhini[2]

[*1]Department of Computer Applications, School of IT, Madurai Kamaraj University, Madurai, India
[2]Technical Support Engineer, Concentrix India Pvt Ltd, Chennai, India

DOI: 10.5281/zenodo.1135392

ABSTRACT

The landscape of distributed computing systems has changed many times over the previous decades. Modern real-world distributed systems consist of clusters, grids, clouds, desktop grids, and mobile devices. Writing applications for such systems has become increasingly difficult. The aim of the Ibis project is to drastically simplify the programming of such applications. The Ibis philosophy is that real- world distributed applications should be developed and compiled on a local workstation, and simply be launched from there. The Ibis project studies several fundamental problems of distributed computing hand-in-hand with major applications, and integrates the various solutions in one programming system.

KEYWORDS: Distributed Computing, IBIS, Multimedia, javagat, Zorilla.

I. INTRODUCTION

The past two decades have seen tremendous progress in the application of high-performance and distributed computer systems in science and industry. Among the most widely used systems are commodity compute clusters, large-scale grid systems, and, more recently, economically driven computational clouds and mobile systems. In the last few years, researchers have intensively studied such systems with the goal of providing transparent and efficient computing, even on a worldwide scale.[1] Unfortunately, current practice shows that this goal remains out of reach.[2] For example, today's grid systems are mostly exploited to run coarse-grained parameter-sweep or master-worker programs. For more complex applications, grid usage is generally limited to straightforward scheduling systems that select a single site for execution. This is unfortunate, as many scientific and industrial applications— including astronomy, multimedia, medical imaging, and bio banking— would benefit from distributed compute resources. Optical networking advances also enable a much larger class of applications to run efficiently on such distributed systems [3]. In addition, research hasn't adequately addressed the problems that can arise from combining multiple unrelated systems to perform a single distributed computation. This is a likely scenario, as many scientific users have access to a wide variety of systems.

Applications that drive the Ibis project include scientific ones (in the context of the VL-e project), multimedia content analysis, distributed reasoning, and many others. The fundamental problems we study include performance, heterogeneity, malleability, fault-tolerance, and connectivity. Solutions to these fundamental problems are integrated in the Ibis programming system, which is written entirely in Java. Examples are: runtime systems for many programming models (divide- and-conquer, master-worker, etc.), a Java-centric communication library (IPL), a library that automatically solves connectivity problems (Smart Sockets), a graphical deployment system, a library that hides the underlying middle wares (JavaGAT), and a simple peer-to-peer middleware (Zorilla). The resulting system has been used in several award-winning competitions, including SCALE 2008 (at CCgrid'08) and DACH 2008 (at Cluster/Grid'08).

The talk describes the rationale behind Ibis, the main applications (in particular multimedia content analysis), and its most important software components. Next, it discusses experiments where Ibis is used to run parallel applications efficiently on a large-scale distributed system consisting of several clusters, a desktop grid, and an Amazon cloud. Finally, the talk discusses recent work in using Ibis for distributed applications that run on mobile devices (smart phones) or that deploy distributed applications from a mobile device. An example is given where a distributed object- recognition application is launched from an Android phone, so the user can

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

perform tasks that would be impossible on the limited resources of the phone itself. The reminder of this paper is divided as follows: In section 2, Real-world distributed computing. In section 3, IBIS programming system. In section 4, multimedia content analysis. In section 5, IBIS deployment system. In section 6 conclusions.

II. REAL-WORLD DISTRIBUTEDCOMPUTING

An ad hoc collection of compute resources that communicate with one another via some network connection constitutes a real-world distributed system, writing applications for such systems is notoriously difficult, as application programmers must take into account all of the problems described above. Deploying the applications is equally hard because each site is likely to have its own middleware and access policies. The uptake of high-performance distributed computing can be enhanced if these complexities are abstracted away by a single software system that applies to any real- world distributed system. Conceptually, such a system should offer two logically independent subsystems: a programming system, offering functionality traditionally associated with programming languages and communication libraries; and a deployment system, offering functionality associated with operating systems. The programming system should allow applications to be not only efficient but also robust by providing programming models that offer support for fault tolerance and malleability— adding and removing machines on the fly—and that automatically circumvent any connectivity problems. The deployment system should allow for easy deployment and management of applications, irrespective of to serve the vast majority of users, ranging from system and application-level software developers to application users, the subsystems should follow a layered approach, with programming interfaces defined at different abstraction levels, each tailored to different users' needs.

III. IBIS PROGRAMMINGSYSTEM

The Ibis programming system provides many programming models, all implemented on top of theIPL.

IPL

The IPL is a Java-based communication library that provides robust communication and resource- tracking mechanisms. It typically ships with the application as jar (Java archive) files, so no additional preinstalled libraries need be present at any destination machine.

The library provides a range of communication primitives including those for point-to-point and multicast communication. It supports streaming communication, which is especially important in high-latency environments as this allows overlapping of serialization, communication, and deserialization of data. The IPL avoids copying overhead as much as possible and uses byte code rewriting to generate efficient serialization and deserialization functions. Consequently, it can significantly outperform Sun remote method invocation (RMI) communication and even C/MPI, in particular for complex data structures.

The IPL has been designed specifically for real- world distributed environments where resources can be added or removed dynamically. It incorporates a mechanism, Join- Elect Leave (JEL), that tracks which resources are being used and what roles they have. JEL is based on the concept of signaling: it notifies the application or runtime system when resources are added to or removed from the computation. To select resources with a special role, it includes elections. JEL thus provides the building blocks for fault tolerance and malleability by giving an up-to-date view of available resources, allowing applications and runtime systems to respond to changes when required. A number of Pure-Java IPL implementations are available using the Ibis Smart Sockets library, TCP (Transmission Control Protocol), UDP (User Datagram Protocol), and Bluetooth. In addition, we provide implementations using specialized non-Java libraries, such as MX (Myrinet) or MPI. The Smart Sockets, TCP, and UDP implementations also work on the Android smart phone platform.

Smart Sockets

Running a parallel application on distributed resources is complicated due to connectivity problems that make direct communication difficult or impossible. Incoming traffic at a node may be restricted by a firewall or NAT. The presence of multiple network interfaces and IP addresses can cause addressing problems, as can private network addresses. The Smart Sockets library automatically solves such problems using existing and novel solutions, including reverse connection setup, overlay routing, and Secure Shell (SSH) tunneling. Smart Sockets creates an overlay network with a set of interconnected support processes called *hubs*. Employing

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

gossiping techniques, the hubs discover which other hubs they can connect to, using SSH tunneling if necessary. This overlay network can be used to help solve connectivity problems or route the application's network traffic. When creating a connection, Smart Sockets initially tries to set up a regular (direct) TCP connection. If this fails, it uses the overlay network to send a request for a reverse connection setup to the target machine. If this also fails, the library creates a virtual connection that uses overlay routing. Smart Sockets also handles machines with multiple network addresses via multi homing.During connection setup, it considers all source and target addresses.

It uses heuristics to determine the combinations most likely to work and then tries these first. Extra identity checks in the protocol ensure that it reaches the correct target machine. A worldwide experiment demonstrated the library's effectiveness: in 30 realistic connectivity scenarios, Smart Sockets was always capable of establishing a connection, while traditional sockets only worked in six ofthese.

Ibis programming models

Besides the IPL, Ibis provides a range of programming models, from low-level message passing to high-level divide-and-conquer parallelism. It implements the following programming models using the IPL:

- MPJ, the MPI binding for Java;
- Satin, a divide-and-conquermodel;6
- RMI, object-oriented remote procedure call;
- group method invocation (GMI), a generalization of RMI to group communication, including multicast and
- all-to-all communication;
- Maestro, a fault-tolerant and self-optimizing dataflow model;7and
- Jorus, a programming model for data-parallel multimedia applications.8
- Challenge at CCGrid 2008 (for scalability), the International Data Analysis Challenge for Finding Supernovae at IEEE Cluster/Grid 2008 (for speed and fault tolerance), and the Billion Triples Challenge at the 2008 International Semantic Web Conference (for general innovation).

IV. MULTIMEDIA CONTENTANALYSIS

We illustrate Ibis with an application that performs real time recognition of everyday objects. Images produced by a camera are processed by an advanced algorithm that extracts *feature vectors* from the video data, which describe local properties like color and shape. To recognize an object, the application compares the object's feature vectors to ones stored earlier and annotated with a name. As this is a compute-intensive problem with soft real time constraints, a large distributed system performs the analysis.8 A data-parallel application running on a single site processes a single video frame. Calculations over consecutive frames are distributed over different sites in a task-parallel manner. The initial application was written in C++/MPI, using TCP and SSH tunnels for wide- area communication. This program used manual deployment, was vulnerable to connectivity problems and partial failures (each causing the entire application to fail), and was frustrating to write and maintain on heterogeneous hardware and middleware. Step by step, we replaced all its components with an implementation in Java and Ibis. With the new program, the Ibis Deploy GUI makes deployment easy, Smart Sockets automatically corrects the connectivity problems, and the JavaGAT accommodates the middleware heterogeneity. We provided robustness by adding fault tolerance and malleability support to the application using the IPL-provided mechanisms. The resulting application is compiled on a desktop machine and easily deployed onto a distributed system. It concurrently uses up to 20 clusters, commonly employing a total of 500 to 800 cores, and a mix of different middleware. The application even runs on the Android smart phone platform, allowing distributed object recognition from mobile devices.

V. IBIS DEPLOYMENTSYSTEM

The Ibis deployment system consists of a software stack and a graphical user interface for deploying and managing applications. The GUI is implemented on top of the JavaGAT.

JavaGAT

Writing distributed applications using existing middleware APIs is a daunting task. APIs change frequently and are often incomplete and too low- level.9 The JavaGAT provides a high-level API that facilitates development of complex applications. This API is object oriented and offers high-level primitives for access to the distributed system, independent of the middleware that implements this functionality. The primitives

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

provide access to remote data, start remote jobs, and support monitoring, steering, user authentication, resource management, and storing of application-specific data. The JavaGAT uses an extensible architecture, wherein *adaptors* (plugins) provide access to the different types of middleware. The JavaGAT also uses *intelligent dispatching* to integrate multiple middleware systems with different and incomplete functionality into a single, consistent system. This technique dynamically forwards application calls on the JavaGAT API to one or more middleware adaptors that implement the requested functionality. Selection occurs at runtime and uses policies and heuristics that automatically select the best available middleware, enhancing portability. If an operation fails, the intelligent-dispatching feature will automatically select and dispatch the API call to an alternative middleware. This process continues untila middleware successfully performs the requested operation. Although such flexibility comes at the cost of some runtime overhead, this is often negligible compared to the cost of the operations themselves. For instance, a Globus job submission takes several seconds, while the overhead introduced by the JavaGAT is less than 10 ms. However, the additional semantics of the high-level API can introduce some overhead. For instance, if a file is copied, the JavaGAT first checks if the destination already exists or is a directory. These extra checks cost time because they require remote operations. The JavaGAT API isn't the lowest common denominator of the underlying middleware APIs. Instead, the JavaGAT offers rich default functionality and can combine features of multiple middleware layers with its intelligent- dispatching technique.

Zorilla

Most existing middleware APIs lack co scheduling capabilities and don't support fault tolerance and malleability. To overcome these problems, Ibis provides Zorilla, a lightweight P2P middleware that runs on any real-world distributed system. In contrast to traditional middleware, Zorilla has no central components and is easy to set up and maintain. It supports fault tolerance and malleability by implementing all functionality using P2P techniques. If resources used by an application are removed or fail, Zorilla can automatically find replacement resources. It was specifically designed to easily combine resources in multiple administrative domains. To create a resource pool, a Zorilla daemon process must be started on each participating machine. Also, each machine must receive the address of at least one other machine to set up a connection. Jobs can be submitted to Zorilla using the JavaGAT or, alternatively, using a command- line interface. Zorilla then allocates the requested number of resources and schedules the application, taking user-defined requirements like memory size into account. The combination of virtualization and P2P techniques thus makes it very easy to deploy applications with Zorilla.

Ibis Deploy

Many applications use the same deployment process. Therefore, Ibis Deploy provides a simple and generic API and GUI that can automatically perform commonly used deployment scenarios. For example, when a distributed Ibis application is running, Ibis Deploy starts the Smart- Sockets hub network automatically. It also automatically uploads the program codes, libraries, and input files (*prestaging*) and automatically downloads the output files (*poststaging*).

VI. CONCLUSIONS

The Ibis programming subsystem is mostly useful for applications written in Java or languages that compile to Java source code or byte code. Java applications can use non-Java libraries through the Java Native Interface and invoke non-Java executables with the Process. exec method. Theoretically, non-Java applications could also use the IPL through the JNI, but this is complicated. Despite these restrictions, many applications have been programmed on top of Ibis. In addition, the Ibis deployment subsystem has been used to deploy both Java and non-Java applications. We're currently researching how to integrate support for accelerators like GPUs, which requires access to non-Java code. In addition, existing high-level programming models don't cover all application domains, leaving some applications to use the IPL directly because they must address locality optimizations, fault tolerance, or malleability themselves. We're thus developing more flexible runtime support in Ibis for a broader range of high-level programming models. Finally, the interoperability layer (JavaGAT) introduces some runtime overhead. In practice, this overhead is insignificant except for operations that provide additional semantics such as remote error checks. More importantly, the JavaGAT's intelligent-dispatching technique leads to more complex error reporting and debugging if operations fail. Instead of a single error message, the user now gets one error message per middleware layer that the JavaGAT attempted to use. Visual debugging and profiling tools should be developed to help the user address these problems

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

VII. REFERENCES

[1] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," Int'l J. High-Performance Computing Applications, Aug. 2001, pp. 200-222.
[2] D. Butler, "The Petaflop Challenge," Nature, 5 July 2007, pp. 6-7.
[3] K. Verstoep et al., "Experiences with Fine- Grained Distributed Supercomputing on a 10G Testbed," Proc. 2008 8thIEEE Int'l Symp. Cluster Computing and the Grid (CCGrid 08), IEEE CS Press, 2008, pp. 376-383.
[4] R.V. van Nieuwpoort et al., "Ibis: A Flexible and Efficient Java-Based Grid Programming Environment," Concurrency and Computation: Practice and Experience, June 2005, pp.1079-1107.
[5] J. Maassen and H.E. Bal, "SmartSockets: Solving the Connectivity Problems in Grid Computing," Proc. 16th Int'l Symp. High Performance Distributed Computing (HPDC07), ACM Press, 2007, pp. 1-10.
[6] R.V. van Nieuwpoort et al., "Satin: A High- Level and Efficient Grid Programming Model," ACM Trans. Programming Languages and Systems, Mar. 2010, article no.9.
[7] C. van Reeuwijk, "Maestro: A Self-Organizing Peer-to-Peer Dataflow Framework Using Reinforcement Learning," Proc. 18th ACM Int'l Symp. High Performance Distributed Computing (HPDC 09), ACM Press, 2009, pp. 187-196.
[8] F.J. Seinstra et al., "High-Performance Distributed Video Content Analysis with Parallel- Horus," IEEE MultiMedia, Oct. 2007, pp. 64-75.

CITE AN ARTICLE

Gurusamy, V., & Nandhini, K. (n.d.). IBIS: THE NEW ERA FOR DISTRIBUTED COMPUTING. *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY, 7*(1), 61-65.