# Encryption Key Search using Java-based ALiCE Grid

Ameya Virkar

Singapore-MIT Alliance
National University of Singapore
Email: ameya@comp.nus.edu.sg

## ABSTRACT

*Encryption Key Search is a compute-intensive operation that consists of a brute-force search of a particular key in a given key space. Sequential execution time for a 56-bit encryption key search is approximately 200,000 years and therefore it is ideal to execute such operation in a grid environment. ALiCE (Adaptive and scaLable internet-based Computing Engine) is a grid middleware that offers a portable software technology for developing and deploying grid applications and systems. This paper discusses the development of the Encryption Key Search application on ALiCE and also presents the performance evaluation of ALiCE using this application.*

## 1 Introduction

Grid is a dynamic network of computing resources that collaborate as a single operating environment, spanning locations and administrative domains and flexibly supporting dynamically changing computing requirements.

A computational grid infrastructure consists of hardware and software components that provide dependable, consistent, pervasive and inexpensive access to high-end computational capabilities [7]. Projects such as Condor-G [8, 17], SETI@home [22], The Globus Computational Grid [5, 6], Javelin [4, 18], Charlotte [3] and MIT's Bayanihan [20, 20] among many have not only contributed ideas on how grid computing systems can be implemented, but they have also demonstrated the potential of grid computing systems.

Table 1 summarizes the related work in the area of grid computing [19]. Communications Technology (Comm. Tech) refers to the underlying technology that is used to build the system, Programming Language (Prog. Lang) refers to the language that the grid system supports, Platforms represent the operating systems running on the machines, Environment refers to the type of network that the grid system is deployed on (Internet refers to the interconnection of computers through any protocol whereas the World Wide Web refers to the use of the Hyper Text Transfer Protocol via Web Browsers) and lastly the Architecture represents the type of architecture used in the grid system.

ALiCE is a lightweight grid-computing middleware for aggregating computational resources. The programming model offered in ALiCE has enabled development of applications such as Geo-rectification of the earth's satellite images in satellite remote sensing and N-body simulations. This paper provides an overview of the ALiCE architecture, the programming model supported on ALiCE and the development of encryption key search application on ALiCE.

The remainder of this paper is structured as follows. Section 2 describes with the architecture and implementation of ALiCE grid system. The ALiCE programming model has been explained in Section 3. Section 4 discusses about the encryption key search algorithm and also gives the performance evaluation results of the ALiCE system using this application. The conclusion and future work is discussed in Section 5.

## 2 ALiCE Grid System

ALiCE is a portable software technology for developing and deploying general-purpose grid applications and systems. It virtualises computer resources on the Internet/intranet into one computing environment through a platform-independent consumer-producer resource-sharing model, and harnesses idle resources for computation to increase the usable power of existing systems on the network.

| Grid System | Comm. Tech | Prog. Lang | Platforms | Environment | Architecture |
|---|---|---|---|---|---|
| Legion [11, 15] | Sockets | C++, Mentat, Fortran, PVM, MPI | Solaris (SPARC), SGI, Linux (x86, Alpha, DEC), HPUX, RS6000 | Internet | *Tree Structure* (global root node) |
| GLOBE [12] | Java IDL | Java, C | WinNT NT, Unix | Internet | *Tree Structure* (global root node) |
| Javelin [4, 18] | Java Applets, Java RMI | Java | Any Java enabled browser | WWW | *Three-Tier* Clients, Brokers, Hosts |
| Calypso [1] | Sockets | C++ | Solaris, Linux, Win NT | Cluster Computing | *Three-Tier* Clients, Brokers, Hosts |
| Charlotte [3] | Java Applets, Java RMI | Java | Any Java enabled browser | WWW | *Three-Tier* Clients, Brokers, Hosts |
| Knitting Factory [2] | Java Applets, Java RMI | Java | Any Java enabled browser | WWW | *Three-Tier* Clients, Brokers, Hosts |
| Condor [8, 17] | RPC | C | Win NT, Unix | Intranet | *Client-Server* |
| Nile [16] | Java, CORBA | Java | Any Java compatible platform | Wide Area Network | Unknown |
| Bayanihan [20] | HORB using Java Applets | Java | Any Java enabled browser | WWW | *Client-Server* Clients are resources only, they cannot submit jobs |
| Globus [5, 6] | Nexus | Java, MPI | Linux, Solaris | Internet | *Tree Structure* (without global root node) |
| ALiCE SoC, NUS | Jini™, Java RMI, JavaSpaces™ | Java | Any Java compatible platform | Internet & Cluster Computing | *Three-Tier* Clients can be a resource provider, a resource consumer, and both |

**Table 1:** Comparison of Grid Computing Systems

## 2.1 Architecture

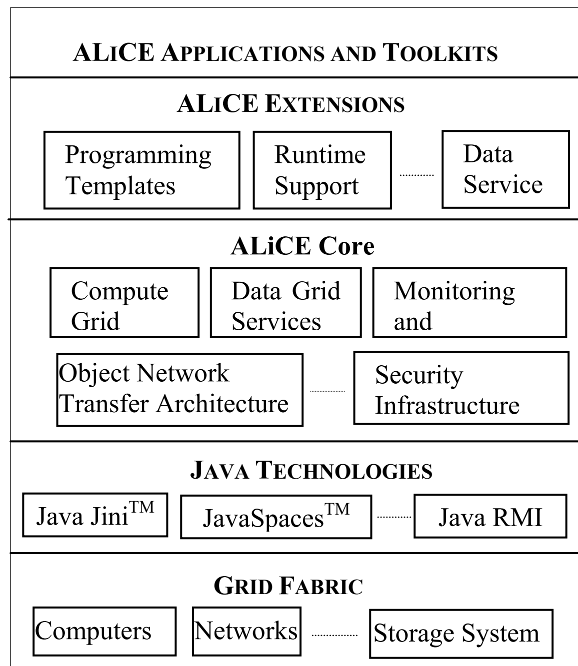


**Figure 1:** ALiCE Architecture

ALiCE layered architecture is shown in Figure 1. The different components at a layer are built on capabilities and behaviors provided by its lower layer.

The grid fabric layer provides the resources that are shared in the grid system. For example, computational resources, storage systems, network resources and sensors are part of the grid fabric layer. Java™ Technologies have been used in developing the ALiCE middleware. Using such technologies, the middleware provides tools that make it easier to build computational grids and grid-based applications.

The layered architecture consists of the following layers:

- *ALiCE Core:* ALiCE Core provides essential *grid services* like resource scheduling, producer management etc., important *data services* like data management and caching, incorporates *object network transfer architecture* (ONTA) to perform communication using the underlying Java technologies, s*ecurity infrastructure* to handle security issues and the *monitoring and accounting system* for performance analysis and billing.
- *ALiCE Extensions:* This layer includes the runtime support system that handles the platform specific information for the program execution. The layer also provides programming templates, built on the ALiCE programming model, to be used for application development. In addition, certain data services are also made available at this layer.

- *ALiCE Applications and Toolkits:* The final layer comprises of the user applications that operate within the ALiCE grid system. Applications are constructed by using the underlying programming templates. The underlying layered structure of the ALiCE architecture is transparent to the application programmer and therefore application development is easy and convenient under the ALiCE grid system.

## 2.2 ALiCE System

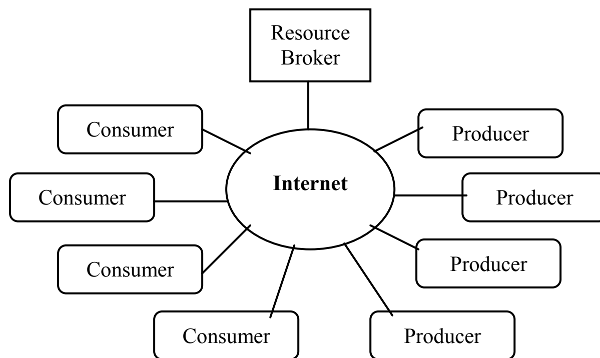


**Figure 2:** ALiCE Components

The main components of ALiCE are outlined below:

- *Consumer:* The consumer submits the applications to the system. The consumer also deals with the collection of results returned by the different tasks of the application. The consumer also includes support to visualize the progress of the execution.
- *Producer:* The producer provides the computer power to run ALiCE applications. The producer receives tasks from the ALiCE system that are dynamically loaded and executed. The results obtained from each task are sent back to the system so that the consumer that has originally submitted the application can receive them.
- *Resource Broker:* The resource broker deals with application scheduling and task scheduling. The resource broker also temporarily stores results returned by producers so that the consumer can collect them.

The ALiCE components interact in the following manner. A consumer machine submits applications to the resource broker. The producers that are willing to volunteer their computing resources get connected to the resource broker. The sub-tasks of the application are generated at the resource broker. The resource broker regulates consumer's resource demand and producer's availability, and dispatches tasks from the task pool for execution at the producers. The producers return the results to the resource broker from where they are sent to the consumers.

## 2.3 Object Communication

In ALiCE grid system, the tasks and messages are communicated with the different nodes using persistent objects. To support transfer of persistent objects over a network, ALiCE includes ONTA (Object Network Transfer Architecture). ONTA offers general APIs to save the state of a live object or class as serialized object, combine it with other classes in an archive file and then load it back at the other node. ONTA uses a generic way to transport the serialized objects over the network using a protocol model, which is as general as possible. ONTA also handles adding of new protocols in the system. ONTA mechanism is shown in Figure 3.

There are six components inside ONTA: *Object Writer*, *Object Repository*, *Remote Object Loader*, *Object Loader*, *File Manager* and *Protocols.* For object communication, The Object Writer serializes the object (1) and creates a jar file containing the object and all classes required by the object. *Object Repository* stores the jar archive and advertises it to be downloaded by remote object loaders (2). *Remote Object Loader* retrieves the jar file reference (3) and downloads the file (4). *Object Loader* restores the serialized object from the jar file (5); *File Manager* handles file naming and storage on the local disk. *Protocols* represent the rules determining the format and transmission of object.

## 2.4 Implementation

ALiCE is implemented in Java$^{TM}$, for full *cross-platform portability*, *extensibility* and *scalability* and uses GigaSpaces$^{TM}$ for communication and resource discovery. GigaSpaces$^{TM}$ is a commercial implementation of Sun Microsystems' Jini$^{TM}$ and JavaSpaces$^{TM}$ technologies. JavaSpaces$^{TM}$ provides a logical distributed-shared memory whereas GigaSpaces$^{TM}$ implements a distributed-shared memory by coupling spaces hosted at different machines.

# 3 Programming in ALiCE

A grid environment is inherently parallel, distributed, heterogeneous and dynamic [14]. Due to such differences, programming applications in a grid environment is beyond the scope of many existing programming tools. It is therefore necessary to build

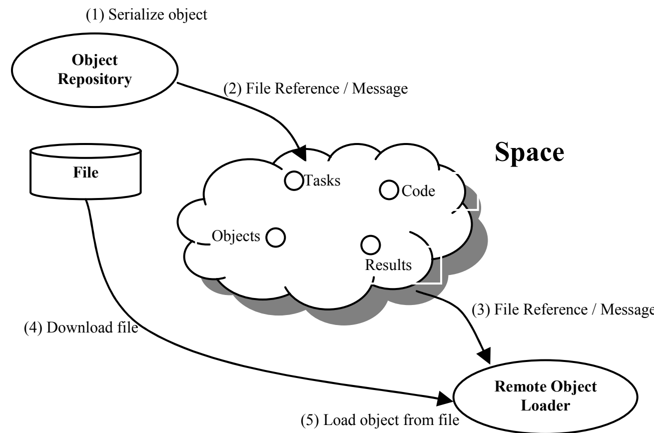


**Figure 3:** Object Communication in ONTA

an effective programming model to support development of applications on ALiCE grid system and thereby support both sequential and parallel computer applications to maximize computer throughput. Similar to typical parallel computer applications, ALiCE programming model allows breaking down large computations into smaller tasks that can be distributed among producers tied to a network to exploit parallelism and speedup.

ALiCE follows the *TaskGenerator-ResourceCollector* programming model. In this model, a consumer submits an application in the form of its Java Archive (JAR) file. The TaskGenerator at the Resource Broker initiates the application and generates a pool of tasks. The tasks are allocated for execution to the Producers. The results of the computation at the Producers are returned to the Resource Broker in the form of a Result object. ResultCollector is initiated at the consumer to support visualization and monitoring of data. The ResultCollector collects Result objects from the resource Broker and uses them for visualization. All the results collected by the Resource Broker are returned to the consumer as a file.

To support the various operations described in the programming model, programming templates have been devised to help in the development of new applications under ALiCE. The programming template has been developed in a way that programmers can exploit the distributed nature of the ALiCE system. The programming templates include methods for sending tasks to ALiCE and also to retrieve the Result objects from ALiCE.

The templates included in the programming model are:

- *TaskGenerator:* TaskGenerator template allows applications to be invoked at the resource broker. It provides methods that allow applications to send tasks to ALiCE system.
- *ResultCollector:* ResultCollector template includes methods that can be invoked at the consumer. The methods allow the application to retrieve results arriving at the consumer.
- *Task:* A Task template is used to specify functions that are needed to execute jobs at the producer. The template also allows the producer nodes to return Result objects to the resource broker upon completing the execution.

# 4 Encryption Key Search

The encryption key search is a method that uses brute force approach to search encryption keys in a given key space. The search is performed to identify one particular key that was used to encrypt the text. An encryption key search application requires immense computational power for searching the key and is therefore suitable to be executed in a grid environment.

Data Encryption Standard (DES) [22, 24] is the most popular algorithm used for encryption of text. A DES key consists of 64 bits of which 56 bits are randomly generated and used directly by the algorithm and the remaining 8 bits are used for error detection. Many projects have been implemented in the past that perform key search in a DES key space. Some projects have used volunteers throughout the Internet to systematically explore the key space. In other projects, expensive special purpose computers were

used to identify an encryption key. Others have built special hardware for processing the keys.

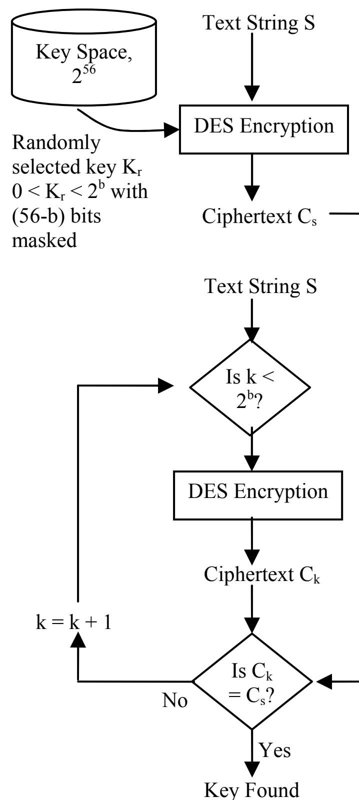


**Figure 4:** Algorithm

In our implementation, a randomly selected key $K_r$ is used to encrypt a known text string S into a ciphertext $C_s$. To identify key $K_r$, every key k in the key space is used to encrypt the same known text string to get ciphertext $C_k$. If for any given key $C_k$ matches with $C_s$, then the brute force algorithm converges and the key value is returned.

Since DES key space is very huge ($2^{56}$ keys), we have modified the application such that it is possible to specify the range of the key space. The bit-length (b) is used to fix largest key in the new key space. The new key space would be of size ($2^b$) keys. In order to use the DES algorithm, the rest of the bits (56-b) of the key are masked. The randomly selected key $K_r$ is also chosen from within the restricted key space ($2^b$ keys) and its remaining bits are masked. Changing the bit-length (b) of the largest key can thus vary the problem size. The algorithm is shown in Figure 4.

The task generator in ALiCE breaks down the key space into smaller tasks and the resource broker then assigns these tasks to the producers. The producers perform the search on the smaller tasks and the results are returned to the consumer. The programming templates are shown in Figure 5.

# 5 Experiments

The performance of ALiCE was tested with the Key Search application. The performance was measured by recording the time required to search the entire key space for a given problem size. The test environment consists of a homogenous cluster connected with Myrinet fast network, and a fast heterogeneous cluster with Ethernet network. All nodes in the clusters are running Red Hat Linux.

The homogenous cluster (Cluster I) is made of 64-nodes where each node is 1.4GHz Intel Xeon dual processor with 1GB of memory. The heterogeneous cluster (Cluster II) consists of 24 nodes of which 16 nodes are Intel Pentium II 400MHz with 256MB of memory and 8 are Intel Pentium III 866MHz with 256MB of memory.

GigaSpaces™ Platform 2.0 has been used in the development and testing. The resource broker, producers and consumer implement Java™ 2 Runtime Environment Standard Edition with Jini™ Starter Kit version 1.2. The Java™ HotSpot Server Virtual Machines are used at the resource broker and producer nodes. The consumer nodes make use of the Java™ HotSpot Client Virtual Machine.

## 5.1 Problem Characteristics

We first measured the time required by a single machine to perform the search. It was observed that the sequential running time for the key search application increases exponentially with increase in the key space size. Therefore in order to compute the time for larger space sizes, we compute the running time on single machine for smaller space sizes and extrapolate the values to compute the time for larger space sizes. The smaller size, which is chosen for this purpose, is same as the size of individual task in the parallel execution. The actual and extrapolated running timings are given in Table 2. The table shows that the extrapolated timings match with the actual running timings on a single machine.

Next we record the extrapolated total running time on a single machine by varying the task size. Using the total key space size, we can estimate the number of tasks that would be created by choosing the particular task size and thereby compute the time taken for running each task. Table 3 shows the values for different task sizes. The table helps in choosing an appropriate task size such that the computation time on every node is larger than the network latency.

| Task Generator | Task |
|---|---|
| | |

```
import alice.consumer.*;
import alice.data.*;
public class MyTaskGenerator extends TaskGenerator {
  public TASKGEN_CLASSNAME() {}

  public void generateTasks() {
    for (long i=0; i<tasks; i++) {
          cipher.encrypt(in,out,8);
          start = i*keysPerTask;
          end = start+keysPerTask-1;
          t = new MyTask(i,start,end,in,out);
          process(t);
    }
  }

  /* Main method - entry point */
  public void main(String args[]) {
    // This is where the tasks are generated, usually in a loop
    this.generateTasks();
  }
}
```

```
import alice.consumer.*;
import java.io.*;

public class MyTask extends Task {
          private long id;
          private long keyStart, keyEnd;
          byte[] byteIn, byteOut;

          public MyTask () {
          }

          public MyTask(long i, long k1, long k2, byte[] in, byte[] out) {
                    id = i;
                    keyStart = k1; keyEnd = k2;
                    byteIn = in; byteOut = out;
          }

          public Object execute () {
               // Test keys from keyStart to keyEnd and return the result
          }
}
```

**Result**

```
import java.io.*;

public class Result implements Serializable {
  public Long key;
  public Boolean found;
  public Long id;
  public long time;

  public Result() {
    key = null;
    found = Boolean.FALSE;
  }

  public void found(Long k) {
    found = Boolean.TRUE;
    key = k;
  }
}
```

**ResultCollector**

```
import alice.result.*;
public class DesCrackerRC extends ResultCollector implements Runnable {


  public RESCOL_CLASSNAME() {
  }

  public void collect() {
    for (long i=0; i<tasks; i++) {
        while (parent.getResultsNoReady() == 0 && !ended)
            ;
        if (ended)  return;
        res = (Result)parent.collectResult();
        keysSoFar += keysPerTask;
    }
    if (res.found.booleanValue()) ended=true;
}
```

**Figure 5:** Snapshot of the Source Code

| Key Length (Bits) | Cluster I | | Cluster II | |
|---|---|---|---|---|
| | Actual | Estimated | Actual | Estimated |
| 24 | 1 min | 1 min | 1 min | 1 min |
| 28 | 18 min | 18 min | 29 min | 29 min |
| 32 | 4 hr 52 min | 4 hr 57 min | 7hr 55 min | 7 hr 57 min |
| 36 | - | 3 days | - | 6 days |
| 40 | - | 3 yrs | - | 5 yrs |
| 56 | - | 201396 yrs | - | 328690 yrs |

**Table 2:** Sequential Execution Time

## 5.2 Distributed Execution

Task size of 50,000,000 keys per task was chosen and the experiments were conducted on 36-bit keys using Cluster I and 32-bit on keys using Cluster II. The key search application was executed on different number of nodes – 4, 8, 10, 12, 16 and 32. The results of the experiment are presented in the table 4. The results reveal the advantages of executing the application in a grid environment. The effective time for key search reduces with increasing number of nodes thereby giving greater speedup.

We now demonstrate the speedup achieved after running the experiments on a single machine and on a cluster of machines. In cluster environment where all processors are homogeneous, both in hardware and in software configuration, workload on these processors does not vary. In such a setting speedup is defined as $T_s/T_p$, where Ts is the time to run the program on one processor and $T_p$ is the time to run the same program on p processors. The speedup obtained by running the search application on 36-bit keys is shown in the Figure 6. We have chosen 50,000,000 keys per task as the task size for measuring the speedup.

We consider these results highly encouraging, although they need to be evaluated further with different key space sizes and higher numbers of nodes. Also, the effects of using other scheduling schemes in the resource broker need to be measured.

| Keys/Task | 32-bit Key | | | 36-bit Key | | 40-bit Key | |
|---|---|---|---|---|---|---|---|
| | # Tasks | Est. Time/Task (min) | | # Tasks | Est. Time/Task (min) | # Tasks | Est. Time/Task (min) |
| | | Cluster I | Cluster II | | Cluster I | | Cluster I |
| 5,000,000 | 859 | 0.3 | 0.5 | 13,744 | 0.3 | 219,902 | 7.0 |
| 10,000,000 | 429 | 0.7 | 1.0 | 6,872 | 0.7 | 109,951 | 14.0 |
| 30,000,000 | 143 | 2.0 | 3.0 | 2,291 | 2.0 | 36,650 | 43.0 |
| 50,000,000 | 86 | 3.0 | 5.0 | 1,374 | 3.0 | 21,990 | 72.0 |
| 100,000,000 | 43 | 6.0 | 10.0 | 687 | 7.0 | 10,995 | 160.0 |

**Table 3:** Estimated Execution Time for Different Task Sizes

| # Nodes | Cluster I | Cluster II |
|---|---|---|
| | 36-bits Key | 32-bits Key |
| Est. Sequential | 78.38 | 8.72 |
| 4 | 23.60 | 3.72 |
| 8 | 11.01 | 2.12 |
| 10 | 8.57 | 1.70 |
| 12 | 7.35 | 1.43 |
| 16 | 5.18 | 1.12 |
| 32 | 2.48 | - |

**Table 4:** Execution Time (hours) for Varying the Number of Nodes

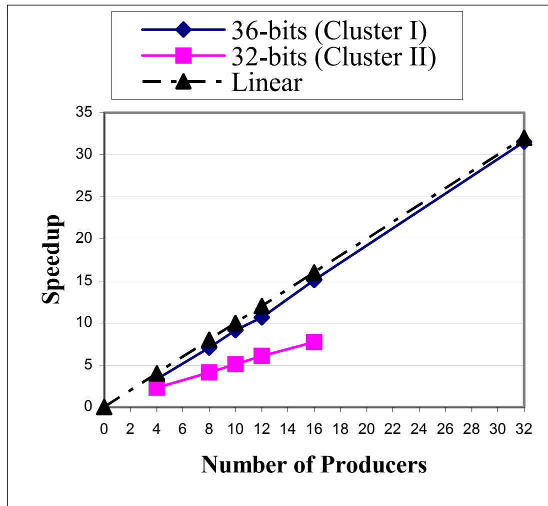


**Figure 6:** Speedup

## 6 Conclusion

The Encryption Key Search application has been developed under the ALiCE grid environment. ALiCE programming templates used for development of this application have helped to exploit the parallelism in the application. Experiments show that speedup of approximately 98% in the execution time has been achieved for a 36-bit encryption key search using 32 nodes. Through the experiments, it has been demonstrated that a grid environment, like ALiCE, can be exploited for compute-intensive problems like encryption key search.

The experiments presented in the paper show the performance of the application on a cluster grid. Future work includes developing a non-java version of the application. Also the application is to be tested in a local-area-network (LAN) and wide-area-network (WAN).

## References


1. Baratloo. A, Dasgupta. P and Kedem, Z. M, Calypso: A novel software system for fault-tolerant parallel processing on distributed platforms, *In Proceedings of 4th IEEE International Symposium on High-Performance Distributed Computing*, 1995
2. Baratloo. A, Karaul. M, Karl. H and Kedem, Z.M, KnittingFactory: An Infrastructure for Distributed Web Applications, *TR1997-748, Department of Computer Science, New York University*, 1997.
3. Baratloo. A, Karaul. M, Kedem. Z and Wyckoff. P, Charlotte: Metacomputing on the Web, *In the Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems*, 1996.
4. Christiansen, B. O, Cappello. P, Ionescu, M. F, Neary, M. O, Schauser, K. E and Wu. D, Javelin: Internet-Based Parallel Computing Using Java, *Concurrency: Practice and Experience*, Volume 9(11), pp. 1139 - 1160, November 1997.
5. Foster. I and Kesselman. C, Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications*, Vol. 11(2), pp 115-128, 1997
6. Foster. I, Kesselman. C and Tuecke. S, The Anatomy of the Grid: Enabling Scalable Virtual Organizations,

*International Journal of Supercomputer Applications*, Volume 15(3), 2001.
7. Foster. I, and Kesselman. C, *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, 1999.
8. Frey. J, Tannenbaum. T, Foster. I, Livny. M and Tuecke. S, Condor-G: A Computation Management Agent for Multi-Institutional Grids, *Journal of Cluster Computing* Volume 5, pp. 237-246, 2002.
9. Germain. C, Néri. V, Fedak. G and Cappello. F, XtremWeb: building an experimental platform for Global Computing, GRID 2000, LNCS 1971, pp. 91-101, Springer-Verlag Berlin Heidelberg, 2000.
10. Grid Computing Info Center (GRID Infoware). http://www.gridcomputing.com
11. Grimshaw, A.S, Lewis, M. J, Ferrari, A. J, Karpovich, J. F, Architectural Support for Extensibility and Autonomy in Wide-Area Distributed Object Systems, *Proceedings of the 2000 Network and Distributed System Security Symposium (NDSS2000)*, February 2000.
12. Homburg. P, M. van Steen, and Tanenbaum, A. S, An Architecture for a Wide Area Distributed System, *In the Proceedings of the Seventh ACM SIGOPS European Workshop*, Connemara, Ireland, pp. 75-82, September 1996.
13. Khunboa, C. and Simon. R, On the Performance of Coordination Spaces for Distributed Agent Systems, *In the Proceedings of the IEEE 34th Annual Simulation Symposium*, April, 2001, Seattle, Washington. pp 7-14, 2001.
14. Lee, Matsuoka, Talia, Sossman, Karonis, Allen and Thomas, A Grid Programming Primer Programming Models Working Group, Grid Forum 1, Amsterdam, 2001.
15. Lewis. M, Grimshaw. A, The Core Legion Object Model, *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, August 1996.
16. Marzullo. K., Ogg. M, Ricciardi. A, Amoroso. A, Calkins. F and Rothfus. E, NILE: Wide area computing for high energy physics, In the Proceedings of 1996 SIGOPS Conference, New York, 1996.
17. Mutka. M and Livny. M, The Available Capacity of a Privately Owned Workstation Environment, *Performance Evaluation*, Volume 12, no. 4 pp. 269-284, July 1991.
18. Neary, M. O, Phipps. A and Richman. S, Javelin 2.0: Java-Based Parallel Computing on the Internet, *Proceedings of Euro-Par 2000*, Munich, Germany, 2000.
19. Prawira Johan, ALiCE: Java-based Grid Computing System, Honours Thesis, School of Computing, National University of Singapore, 2002.
20. Sarmenta. L.F.G, Bayanihan: Web-Based Volunteer Computing Using Java, *In the Proceedings of `the 2nd International Conference on World-Wide-Computing and its Applications (WWCA'98)*, Tsukuba, Japan, March 1998.
21. Sarmenta. L.F.G, Volunteer Computing, Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT, March 2001.
22. Schneier. B, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd Edition, John Wiley & Sons Publishing Group, pp. 265-302, 1996.
23. SETI@home: Search for Extraterrestrial Intelligence at Home. http://setiathome.ssl.berkeley.edu/
24. Stinson. D R, *Cryptography: Theory and Practice*, 2nd Edition, Chapman & Hall Publishing Group, pp. 95-101, 2002.