Boosting Geographic Information System's Performance using In-Memory Data Grid

Barkha Bahl[1], Vandana Sharma[2] and **Navin Rajpal[3]**

Submitted in April 2012; Accepted in July 2012

Abstract - A typical Geographic Information System(GIS) is information system that integrates, stores, edits, analyzes, shares and displays geographic information for effective decision making. The focus here is to refine the storing and retrieving capabilities of any GIS. GIS application have a very high performance and scalability requirement, such as query response time of less than 3 seconds, 120000 customer sessions per hour and 100000 data addition/updates per day. Also an ideal GIS application always deal with high concurrent load, frequent database access for mostly read only data, and non-linear growth of mostly read only data over period of time. These all are the factors which lead to performance impact in the application. This research proceeds to understand how the In-Memory Data-Grid solution is better than other solutions and how can it be leveraged to implement a very high performing and highly scalable GIS applications.

Index Terms - ***In-memory data grid, Cache memory, Geographic Information system (GIS), Distributed cache***

1. INTRODUCTION

Geographic Information system, commonly known as GIS is a computer system capable of capturing, storing, analyzing, and displaying geographically referenced information, that is, data identified according to location. Practitioners also define a GIS as including the procedures, operating personnel, and spatial data that go into the system.

A GIS application[7] requires low response time, very high throughput, predictable scalability, continuous availability and information reliability which can be provided by In-Memory Data Grid.

In-Memory Data Grid is a Data Grid that stores the information in memory in order to achieve very high performance, and uses redundancy - by keeping copies of that information synchronized across multiple servers in order to ensure the resiliency of the system and the availability of the data in the event of server failure[5].

Over the last few years, In-Memory Data Grids have become an increasingly popular way to solve many of the problems related to performance and scalability, while improving availability of the system at the same time. In-Memory Data Grid allows eliminating *single points of failure* and *single points of bottleneck* in the application by distributing the application's objects and related processing across multiple physical servers.

One of the easiest way to improve application's performance is to bring data closer to the application, and keep it in a format that the application can consume more easily.

Most enterprise applications are written in one of the object-oriented languages, such as Java or C#, while most data is stored in relational databases, such as Oracle, MySql or SQL Server. This means that in order to use the data, the application needs to load it from the database and convert it into objects. Because of the impedance mismatch between tabular data in the database and objects in memory, this conversion process is not always simple and introduces some overhead, even when sophisticated O-R mapping tools, such as Hibernate or Eclipse Link are used.

Caching objects in the application tier minimizes this performance overhead by avoiding un-necessary trips to the database and data conversion. This is why all production-quality O-R mapping tools cache objects internally and short-circuits object lookups by returning cached instances instead, whenever possible.

2. PROBLEM STATEMENT

2.1 Introduction to the Problem

Customer expectations from GIS systems have evolved significantly over a period of time [4]. Today customers are expecting better and faster online experience.

Several architectures are proposed to retrieve necessary, interested and effective information efficiently and at the same time provide scalable platform for GIS application. However, the results of these architectures generally become unsatisfactory and prone to performance loss over the period of time. As soon as the customer base increases, the performance starts retarding.

3. PROPOSED SYSTEM

The proposed system is trying to inculcate the technology called distributed cache in a GIS application. This technology will not only boost performance of application but will also provide many more features to it. The first step in our paper is a strong research base of prevalent architectures and secondly an in-depth study of distributed cache technology . After the research we will try to prove our concept through a small proof of concept.

If we are able to incorporate distributed cache in an GIS application the following feature would be achieved

[1]*Research Scholar, GGSIP University, New Delhi*
[2]*Dy.Director General, NIC, New Delhi*
[3]*Professor and Dean, USIT, GGSIP University, New Delhi*
E-mail: [1]*barkha69@rediffmail.com,*
[2]*sharma.vandana@nic.com and* [3]*navin_rajpal@yahoo.com*

1. Low response time
2. High throughput
3. Eliminate bottlenecks
4. Predictable scalability
5. Continuous availability
6. Failover support
7. Information Reliability

4. LITERATURE REVIEW

Simple database retrieval architecture is still the back bone for most of the complex architectures in use today [8]. GIS application generally contains a program running on server, and is connected to a database. Numbers of users are connected to this program to query, update, delete or add different items.

Initially, application was deployed on a server which originally supports 5000(say) users at a time, which means that at a time 5000 users could connect to the server. No matter, how powerful a server was, for sure it would have some limit on number of users it could support, and therefore as an example here we have assumed that server could support 5000 users at a time.

There is further limit on number of users, whose requests required access to database that could be processed simultaneously. The reason behind this was that, connections created to database were generally heavy, as many connections to database at the same time were not feasible. To efficiently use connections to access to database, developers generally used connection pools, and set a limit on number of connections that could be active at a time. Other then performance issues the other issues regarding the architecture were:

4.1. POF, which stands for single point of failures. In the architecture there were three single points of failures: application, database and server. In case either database crashes or server crashes or application crashes, complete application would be down and no one would be able to access the application or use application.

4.2. Shared resources were always performance bottlenecks and greater the number of connections/users a shared resource would have more will be the affect on performance. Whereas in the previous architecture, database was a shared resource, which could not support large amount of users at same time.

4.3. Another reason of low performance with the basic architecture was the step required to convert data stored in database to application object, when user queries for data stored in database, and step required reading application object to store data in database.

To take care of number of users supported by application, load balancer was introduced [2]. Load balancer's responsibility is to distribute the load efficiently among different servers/applications capable of process the request. In this architecture load balancer application is run on one system and GIS application is deployed and run on more than one server which is further connects to single database. Load balancer forwards the user requests to any of the server configured with load balancer based on the load of the server. The use of load balancer tremendously increased the number of users that could simultaneously connect to application, as number of servers running the application was increased. But there were still large performance issues with the architecture [9].

Still, there is limit on number of users whose requests required access to the database, the reason being limit on number of connections that could be made simultaneously to the database.

SPOF still existed. Though server running the application was no longer, single point of failure, as there were more servers which were present, which would be able to keep application running even if any server or application running on any server crashes. This would remain transparent to users, as users were no longer interacting with the server hosting the application, but users were interacting with the load balancer. When load balancer would get news of one of the server being down, it would then exclude that server from its list of active servers and stop delegating any of the user requests to that specific server. But database was still single point of failure, as we were using single database, and if that database would crash, the application would fail.

Cris J. Holdorph[3] gave an approach to work with distributed database instead of single database. In this scheme, it was considered that each server which was connected to a load balancer was having its own database.

Though, number of users which could be supported now increased, compared to above discussed schemes, but this scheme would require another extra process to replicate the data stored in one database to other databases. This was required to take care of scenario, when user requests were sent to different database. The result sent back should be consistent and independent of data stored on database.

Jim Handy[6] defined a scheme in which multiple servers were connected to single cache, which are further connected to the database.

The number of connections that could be made increased (though this number depends on the server on which cache is hosted). Also the read queries would be much faster, and performance of write queries to the database would be improved if the updates were done in cache synchronously, and asynchronously saved in database by some other process. But there were still some disadvantages related to this scheme like cache and database were still single point of failure, if any of it crashed, application would not be available. Data-intensive queries would run on complete data in cache, which was not very efficient.

In recent past there was a concept of In-Memory Data Grid and related products which have become famous, which could be used to improve performance of applications which are highly affected by database operations and mostly read only operations [8]. In GIS applications most of the requests are related to read-only requests which require reading something from database. Most of the users request sent to server are read-only request and insert/update command is used only when new point is located.

Paul Colmer[5] described the features provided by In-memory

Data grid, which makes it a good choice for GIS application. An In-Memory Data Grid achieves low response time for data access by keeping the information in-memory and in the application object form, and by sharing that information across multiple servers. In other words, applications may be able to access the information that they require without any data transformation step .

Performance is further improved by coalesces multiple changes to a single application object and batches multiple modified application objects into a single database transaction, meaning that a hundred different changes to each of a hundred different application objects could be persisted to a database in a single, large and thus highly efficient transaction[10].

Arindam Chakravorty[1] discussed various topologies in which cache could be used to overcome the limitations of above schemes**.** In-Memory Data Grid supports three types of caches. These are Distributed, Near and Replicated cache topology.

Distributed cache is one in which each node in the server contains a unique set of application data in the cache. To scale the capacity of cache, increase the nodes in the cluster. Any type of cache will involve serialization /de-serialization and network transfers for application data read and write access in the cache. Distributed cache is best when the applications requires heavy volume of read and write application data.

Distributed Cache architecture is shown in Figure 1.

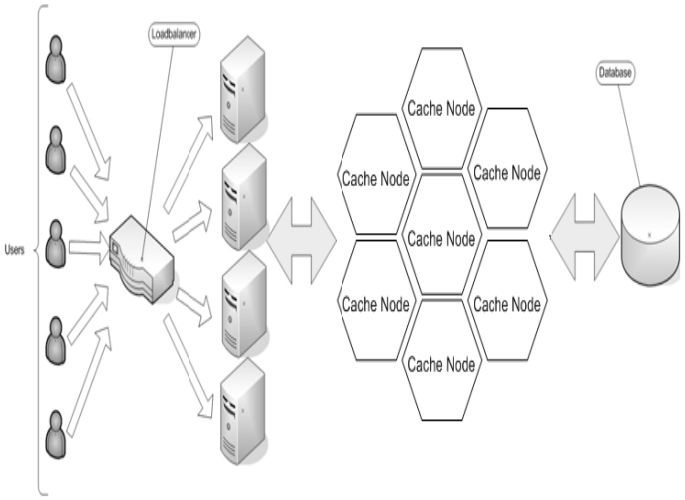

Figure 1: "Distributed Cache Architecture"

Near cache is each client node containing small amount of data in the local cache and larger amount of data in the distributed cache and these caches are synchronized with each other. There is some overhead involved with synchronizing the caches.

In Replicated cache each node in the cluster will contain all the application data in the cache. Replicated cache is best when application requires less application data and highly read access from cache.

5. GRID CLUSTER ARCHITECTURE

5.1 Grid Cluster Architecture

In-Memory Data Grids[8] are built on a fully *clustered* architecture. Grid is based on a peer-to-peer clustering protocol, in which servers are capable of:

5.1.1. **Speaking to Everyone:** When a party enters the conference room, it is able to speak to all other parties in a conference room.

5.1.2. **Listening:** Each party present in the conference room can hear messages that are intended for *everyone*, as well as messages that are intended for that particular party.

5.1.3. **Discovery:** Parties can only communicate by speaking and listening; there are no other senses. Using only these means, the parties must determine exactly who is in the conference room at any given time, and parties must detect when new parties enter the conference room.

5.1.4. **Working Groups and Private Conversations:** Although a party can talk to everyone, once a party is introduced to the other parties in the conference room (i.e. once discovery has completed), the party can communicate directly to any set of parties, or directly to an individual party.

5.1.5. **Death Detection:** Parties in the conference room must quickly detect when parties leave the conference room – or die.

Using the conference room model provides the following benefits:

1. There is no configuration required to add members to a cluster. Any program running grid application when starts will automatically join the cluster and be able to access the caches and other services provided by the cluster. When a program joins the cluster, it is called a *cluster node*, or alternatively, a *cluster member*.
2. Since all cluster members are known, it is possible to provide redundancy within the cluster, such that the death of any one node does not cause any data to be lost.
3. Since the death or departure of a cluster member is automatically and quickly detected, failover occurs very rapidly, and more importantly, it occurs transparently, which means that the application does not have to do any extra work to handle failover.
4. Since all cluster members are known, it is possible to load balance responsibilities across the cluster. Grid does this automatically by distributing the load evenly across cluster. Load balancing automatically occurs to respond to new members joining the cluster, or existing members leaving the cluster.

6. READ-THROUGH CACHING

When an application asks the cache for an entry, for example the key X, and X is not already in the cache, data grid will automatically delegate to the cache-store which is responsible for loading data into cache, and this cache-store will now load X from the underlying datasource.

If X exists in the datasource, the cache-store will load it, return it to data grid, which is then placed in the cache for future use and also data X is returned to the application code that requested it. This is called **Read-Through** caching.

7. WRITE-THROUGH CACHING

Coherence can handle updates to the datasource in two distinct ways, the first being Write-Through[2].

In this case, when the application updates a piece of data in the cache the operation will not complete (i.e. the put will not return) until data is also persisted to the underlying datasource. This does not improve write performance at all, since the user is still dealing with the latency of the write to the datasource[10].

8. REFRESH-AHEAD CACHING

In the Refresh-Ahead scenario, Coherence allows a developer to configure the cache to automatically and asynchronously reload (refresh) any recently accessed cache entry from the cache loader prior to its expiration.

The result is that once a frequently accessed entry has entered the cache, the application will not feel the impact of a read against a potentially slow cache store when the entry is reloaded due to expiration. The refresh-ahead time is configured as a percentage of the entry's expiration time; for instance, if specified as 0.75, an entry with a one minute expiration time that is accessed within fifteen seconds of its expiration will be scheduled for an asynchronous reload from the cache store.

9. WRITE BEHIND CACHING

In the Write-Behind scenario, modified cache entries are asynchronously written to the datasource after a configurable delay, whether after 10 seconds, 20 minutes, a day or even a week or longer.

For Write-Behind caching, grid generally maintains a write-behind queue or any data structure which stores the data that needs to be updated in the datasource. When the application updates X in the cache, X is added to the write-behind queue (if it isn't there already; otherwise, it is replaced), and after the specified write-behind delay data grid service will update the underlying datasource with the latest state of X.

Note that the write-behind delay is relative to the first of a series of modifications – in other words, the data in the datasource will never lag behind the cache by more than the write-behind delay.

The result is a "read-once and write at a configurable interval" (i.e. much less often) scenario. There are four main benefits to this type of architecture:

1. The application improves in performance, because the user does not have to wait for data to be written to the underlying datasource.
2. The application experiences drastically reduced database load: Since the amount of both read and write operations is reduced, so is the database load. The reads are reduced by caching, as with any other caching approach. The writes - which are typically much more expensive operations - are often reduced because multiple changes to the same object within the write-behind interval are "coalesced" and only written once to the underlying datasource ("write-coalescing"). Additionally, writes to multiple cache entries may be combined into a single database transaction.
3. The application is somewhat insulated from database failures: the Write-Behind feature can be configured in such a way that a write failure will result in the object being re-queued for write. If the data that the application is using is in the cache, the application can continue operation without the database being up.
4. Linear Scalability: For an application to handle more concurrent users you need only increase the number of nodes in the cluster; the effect on the database in terms of load can be tuned by increasing the write-behind interval.

10. STATISTICS FOR COMPARISON

10.1 Database and In-Memory Data Grid Performance

The comparison shows that when the data is stored conventionally in databases the processing speed is more as compared to when it is stored in cache. The results have been shown in figure 2 and figure 3.

11. CONCLUSION

An effective caching mechanism is the foundation of any distributed-computing architecture. The focus of this article was to understand the importance of caching in designing effective and efficient distributed architecture. In memory data grid method was finally implemented for the same. It has been observed that retrieval time of GIS application's data saved using in memory data grid method is much less as compared to when the data is saved using the conventional database storage method. Thus, the use of distributed cache technology for spatial data storage will boost the performance of GIS application.

FUTURE SCOPE

Object relational mapping is a way to bridge the impedance mismatch between object-oriented programming (OOP) and relational database management systems (RDBMS). Many commercial and open-source ORM implementations are becoming an integral part of the contemporary distributed architecture. ORM technologies are becoming part of the mainstream application design, adding a level of abstraction. Implementing ORM-level cache will improve the performance of a distributed system. Therefore, this method can be used to improve the performance of the GIS application. In future, the digitized data required for GIS application can be stored using proposed Triangular Pyramid Framework for Enhanced object relational vector data model[3] under the distributed cache environment using in memory data grid for better results.

REFERENCES

[1]. Arindam Chakravorty,"Guide to choose best caching strategy for your application", http://www.csqldb.com, pg. no. 2-11, October 2009.

[2]. Aleksandar Seovic, Mark Falco, Patric Peralta, "Designing Coherence Caches", Oracle Coherence 3.5 Book. Packt Publishing. April 2010.

[3]. B. Bahl, N. Rajpal, V. Sharma,"Triangular Pyramid Framework for Enhanced Object relational Dynamic Data model for GIS. IJCSI", International Journal of Computer Science Issues, vol. 8, issue 1, January 2011,ISSN (online):1694-0814,pg. 320-328,2011

[4]. Ed Upchurch & John Murphy , "Why Worry about Performance in E-Commerce Systems", Proc. Of 16th UK IEE Teletraffic Symposium, May 2000.

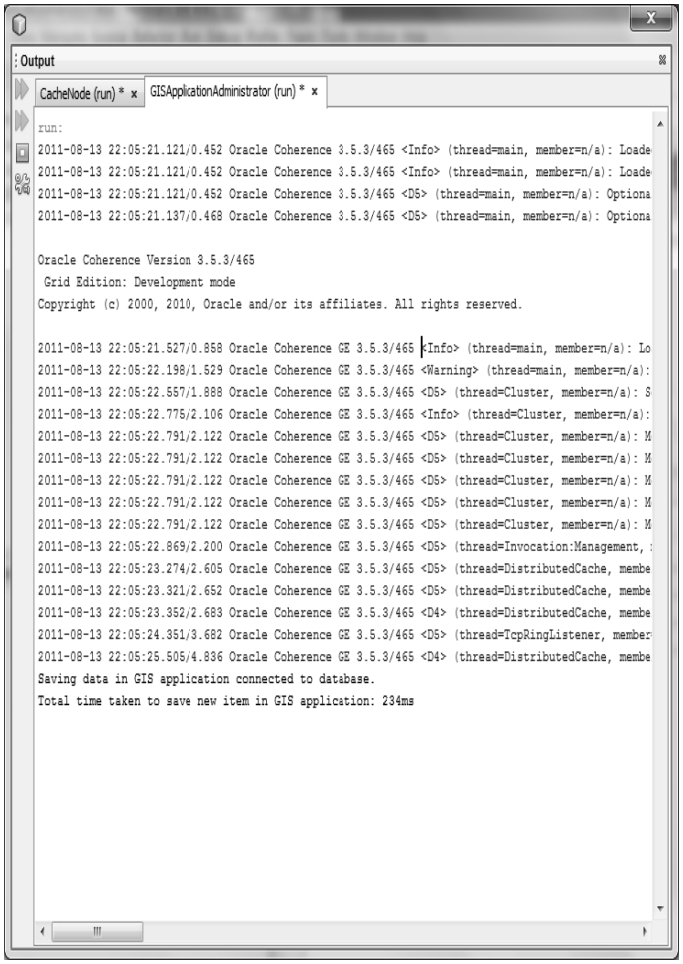

Figure 2: "Saving Item in Databases"

[5]. Paul Colmer, "In-Memory Data Grid for improving performance ", . http://highscalability.com/blog/2011/12/21/in-memory-data-grid-technologies.html, August 2010.

[6]. Jim Handy, "Maintaining Coherency in Caches Systems", The Cache Memory Book. Academic Press, Elsevier Store, Pg. no. 138-141, March 1998.

[7]. Hanan Samet, " Issues, developments, and challenges in spatial databases and geographic information systems(GIS)", GIS'01 proceedings of the ACM International Symposium on Advances in geographic information systems . pg. 1-1, ACM, ISBN -1-58113-443-6., 2001. John Murphy, "Assuring performance in ecommerce system", July 2004.

[8]. Jonathan Purdy, "Data Grids & SOA", An Oracle white paper, Oracle Corporation , pg.-1-12, May 2007

[9]. Rajendaram Sundaram , "Oracle Coherence Cache for a high performance ecommerce application", HCL, 2010.

[10]. Harvendra Kumar, A.K.Verma, "Comparative Study of distributed computing paradigms", BVICAM's International Journal of Information Technology, BIJIT, issue 2, vol. 1 No. 2, July-Dec., 2009.

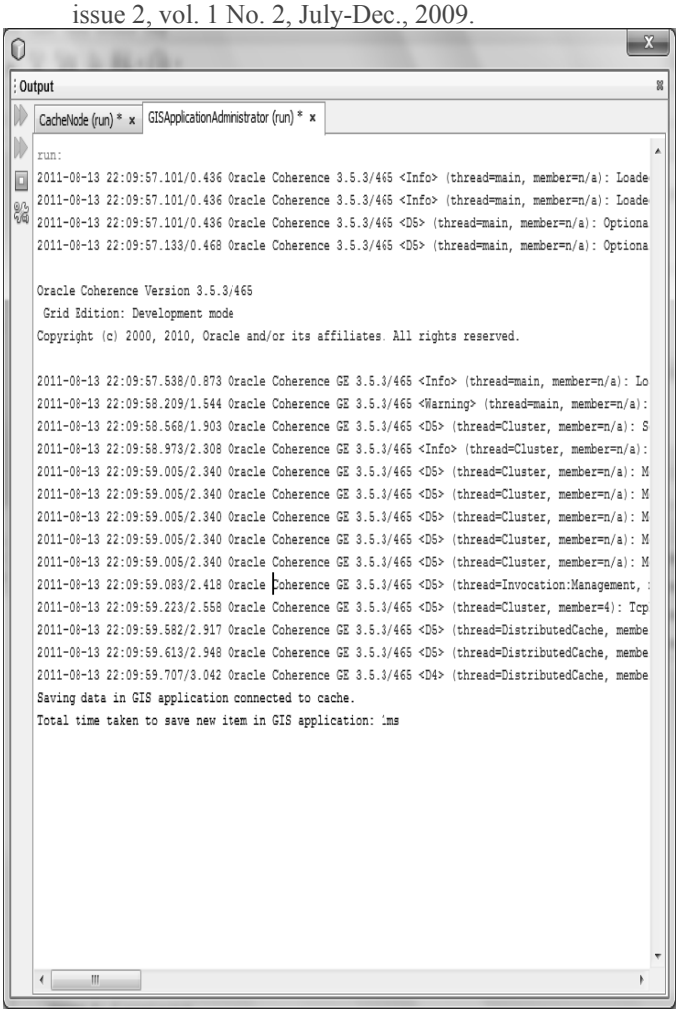

Figure 3: "Saving item in cache"

Continued from page no. 467

[1]. Internet World Stats (2010, June 31). Internet Usage Statistics: The Internet Big Picture (World Internet Users and Population Stats). Retrieved December 21, 2010, from http://www.internetworldstats.com/stats.htm.

[2]. The Economist (2010, July 1). War in the Fifth Domain: Are the Mouse and Keyboard the New Weapons of Conflict? Retrieved June 23, 2012, from http://www.economist.com/node/16478792.

[3]. Techopedia (2012, June 6). Whaling: Phishers look to land a big catch. Retrieved June 23, 2012, from http://www.techopedia.com/2/28617/security/whaling-phishers-look-to-land-a-big-catch .

[4]. MessageLabs Intelligence Report (2010, September 20). MessageLabs Intelligence September 2010. Retrieved December 21, 2010, from http://www.messagelabs.com/mlireport/MLI_2010_09_September_FINAL_EN.PDF.

[5]. Webopedia. Phishing. Retrieved June 23, 2012, from http://www.webopedia.com/TERM/P/phishing.html .

[6]. Trinity College, University of Cambridge. Phishing. Retrieved June 23, 2012, from http://www.trin.cam.ac.uk/index.php?pageid=560.

[7]. Citizens State Bank (2012, March 1). Fraud Busters: How to identify and protect yourself from Internet Fraud. Retrieved June 23, 2012, from www.csbnow.com/fraud-busters.pdf.

[8]. McAfee (2010, March 9). McAfee Inc. Unveils New Consumer Threat Alert Program: A Warning for Consumers about the Most Dangerous Online Threats. Retrieved June 22, 2012, from http://www.mcafee.com/us/about/news/2010/q1/20100309-01.aspx.

[9]. Cabinet Office, U.K. Government (2011, February 17). The Cost of Cyber Crime. Retrieved June 23, 2012, from http://www.cabinetoffice.gov.uk/sites/default/files/resources/the-cost-of-cyber-crime-full-report.pdf.

[10]. The Ethical Hack3r (2010, July 3). [Interview] The Jester. Retrieved June 23, 2012, from http://www.ethicalhack3r.co.uk/security/interview-the-jester/.

[11]. University of Calgary (2009, October 26). Scareware: Don't get spooked!. Retrieved June 23, 2012, from http://www.ucalgary.ca/it/files/it/scareware_poster1_oct_web_1.pdf.

[12]. The Cooperative Association for Internet Data Analysis. AS Rank: Information for a single AS: AS Relationship Table (AS 43513). Retrieved December 21, 2010, from http://bit.ly/as-43513-ranking-by-CAIDA.

[13]. Centix. Threats. Retrieved June 23, 2012, from http://www.centix.be/content.aspx?PageId=135.

[14]. Kusam, Pawanesh Abrol, & Devanand, "Digital Tampering Detection Techniques: A Review", BVICAM's International Journal of Information Technology, BIJIT, issue 2, July-December 2009, Vol. 1 No. 2.

[15]. VirSCAN.org (2010, September 27). EBIH.EXE. Retrieved June 23, 2012, from http://virscan.org/report/5eee80002e0f85f3aec137b9cd30eeda.html.

[16]. B. B. Jayasingh & B. S. Swathi, "A Novel Metric for Detection of Jellyfish Reorder Attack on Adhoc Network" BVICAM's International Journal of Information Technology, BIJIT, issue 3, January-June, 2010 Vol. 2, No. 1.