Optimum off-line trace synchronization of computer clusters

To cite this article: Masoume Jabbarifar et al 2012 J. Phys.: Conf. Ser. 341 012029

View the article online for updates and enhancements.

Related content

- A Parallel Supercomputer Implementation of a Biological Inspired Neural Network and its use for Pattern Recognition
  Vincent de Ladurantaye, Jean Lavoie, Jocelyn Bergeron et al.

- Bioinformatics algorithm based on a parallel implementation of a machine learning approach using transducers
  Abiel Roche-Lima and Ruppa K Thulasiram

- Simulating the ATLAS Distributed Data Management System
  Martin Barisits, Vincent Garonne, Mario Lassnig et al.

Recent citations

- GTSO: Global Trace Synchronization and Ordering Mechanism for Wireless Sensor Network Monitoring Platforms
  Marlon Navia et al

- Online Incremental Clock Synchronization
  Masoume Jabbarifar et al

- LIANA: Live incremental time synchronization of traces for distributed systems analysis
  Masoume Jabbarifar and Michel Dagenais
Optimum off-line trace synchronization of computer clusters

Masoume Jabbarifar, Michel Dagenais, Robert Roy and Alireza Shameli Sendi

Department of Computer and Software Engineering,
École Polytechnique de Montréal, Montréal, Canada

E-mails: {masoume.jabbarifar, michel.dagenais, robert.roy, alireza.shameli-sendi}@polymtl.ca

Abstract. A tracing and monitoring framework produces detailed execution trace files for a system. Each trace file contains events with associated timestamps based on the local clock of their respective system, which are not perfectly synchronized. To monitor all behavior in multi-core distributed systems, a global time reference is required, thus the need for traces synchronization techniques. The synchronization is time consuming when there is a cluster of many computers. In this paper we propose an optimized technique to reduce the total synchronization time. Compared with related techniques that have been used on kernel level traces, this method improves the performance while maintaining a high accuracy. It uses the packet rate and the hop count as two major criteria to focus the computation on more accurate network links during synchronization. These criteria, tested in real-word experiments, were identified as most important features of a network. Furthermore, we present numerical and analytical evaluation results, and compare these with previous methods demonstrating the accuracy and the performance of the method.

1. Introduction

The appearance of multi-core processors in computer clusters represents an evolutionary change in conventional computing to obtain high performance computing [13]. However, these systems may exhibit coherency problems when parallel programs access shared resources, thus creating hard to debug timing related problems. It is therefore crucial to have proper tools to monitor, trace and analyse system execution, in order to identify functional and performance problems. A trace facility aims to keep track of functional flow and report relevant changes at certain times. Global trace analysis, however, faces the problem that the cores of each node of the cluster have their own clock not synchronized with all the others [10]. Dealing with this problem is even more complicated in multi-level tracing (tracing in virtual machines, middleware and applications layer).

The Linux Trace Toolkit next generation (LTTng) [2], developed at École Polytechnique de Montréal, provides a detailed execution trace of the Linux operating system with low overhead. Moreover, LTTng is capable of handling huge traces of several gigabytes or more [16]. However, a new method is required to handle huge traces while allowing the collection of traces from multiple systems and embedded devices, for both on-line and a posteriori off-line analysis and viewing. Furthermore, the user of LTTng expects to see the output of real-time analysis to diagnose possible problems properly. Therefore, LTTng should be able to visualize traces from several distributed systems on a common reference time base.

In a computer cluster, multiple nodes produce separate trace streams independently. There are timestamps on the events of the trace. Since timestamps are recorded based on a logical time that runs.
natively on each node, the logical order of events cannot be guaranteed [6]. Trace synchronization
with due attention to high precision and low intrusiveness has been achieved recently. It complements
trace analysis with a post-processing step called offline synchronization [1]. The objective of this
paper is to optimize the total trace synchronization time for huge computer clusters. The ideal situation
is to improve the total synchronization time while preserving the synchronization accuracy.

This paper is organized as follows: first, we investigate earlier work and several existing
synchronization algorithms. Then, we present and discuss our model in Section 3. In Section 4,
experimental results are presented and results are compared with existing methods. Finally, we
conclude our work by reviewing the significant points of this paper and suggest future work in Section
5.

2. Related Work

This paper focuses on offline synchronization. Several papers have proposed algorithms in this area
[3, 4, 5, 6, 8, 9, 10, 11, 12]. When packets are exchanged between a pair of nodes, the receiving and
sending times will not be shown properly because the clocks of the two nodes are not synchronized
and the time offset should be calculated [9]. Since the packet is necessarily sent before it can be
received, this relationship can be used to help synchronize two nodes. Thus, a conversion function is
needed to convert these two times to a common time.

The first offline time synchronization method was proposed by Duda et al. [3]. He proposed two
synchronization algorithms, Linear Regression and Convex-Hull. They estimate a conversion function
between a pair of clocks. In both algorithms, the conversion function appears as a line. Drift and offset
between the two clocks are extracted from this linear model. In a two dimensional space, based on
times of the nodes A and B, the Linear Regression algorithm tries to map all points on a line; thus,
each point will affect the position of the line. In reality, network latency and such problems between
two nodes cause problematic outlying points which ideally should not affect the delay and offset
computed from the linear regression and should be ignored to increase the accuracy [3, 8]. Convex
Hull is an accurate algorithm that assumes minimum sending times and maximum receiving time; in
this way, it finds the area that has minimum latency and ignores outlying points. Therefore, the
estimated line is more accurate than with Linear Regression.

Khlifi et al. [4] proposed two algorithms named average and direct skew removal techniques. The
second technique is a brute force algorithm which iteratively divides the duration of the trace into next
intervals. It considers the effect of the clock resolution on the accuracy of delay measurements and
provides more accurate synchronization compared with other techniques.

Poirier et al. [1] presented an accurate method for synchronizing distributed traces. This method is
applied to traces that are recorded at the kernel level with low intrusiveness. They apply Convex-Hull
as conversion function to the clocks of traced nodes.

Clement et al. [5] evaluated the system characteristics impact on the trace synchronization
accuracy. Firstly, they studied the tracing duration impact. They proposed to divide long duration
traces into 30 minute segments since the error in the clock drift calculation started to increase
significantly after approximately 45 minutes of tracing, while it was almost stable during the first 30
minutes. The error increases because of the variation of the clock drift with time due to the Allan
deviation [14] and environmental effects on the clock circuit frequency such as temperature and
supply voltage variations. The second parameter studied is the system load impact, when there is a
heavy load on the major subsystems, CPU, memory, and disk. They found that the transmission time
and response time measurement variations, caused by interrupt latency in a loaded system, influence
the clock drift computation directly and subsequently the time synchronization accuracy [5]. The third
parameter studied is hop impact, where there are more than one path between two nodes. In that case,
the offset between the two indirectly linked nodes may be computed by adding the offsets along a
path, from one intermediate hop to the next. The path that has fewer hops generally provides a higher
accuracy. In the case where we have a direct path, it is normally better to choose it to synchronize two
nodes [5].
3. Architecture of Proposed Model

Figure 1 illustrates the architecture of the proposed model. It consists in four connected modules. Each module receives input from one or more modules and sends outputs to other modules. The input of this architecture is fed by LTTng [2] and consists of two or more unsynchronized trace files. The output is two or more synchronized traces. The output format is compatible with LTTV (Linux Trace Toolkit Viewer) [2] which is able to show synchronized traces. The following modules are present in this architecture:

- **Processing module**: traces are gathered from all of nodes distributed in a computer cluster and we have to analyze them. In order to synchronize two nodes, we need network traffic exchanges between them. We have to extract packet exchange events and dispatch those events to the next module. Thus, this module captures network traffic and computer activity and extracts the necessary information for the matching module.

- **Matching module**: event processing feeds the events one by one. However, the Analysis module works on groups of events. Thus, the Matching module is responsible for forming these groups. The relations between the packets are of different types (“one to one”, “one to many”, or a mix) and this will influence the overall behavior of the module for TCP, UDP or MPI. This module must recognize related sent and received packets and group them. For example, for linear regression, we need the round trip time (RTT), so this module makes a group after finding an acknowledgment packet, and the acknowledge time will be assumed as reference time.

- **Analysis module**: as mentioned, there are two methods to synchronize time: Linear Regression and Convex-hull. In this module, the user can chose any of these methods to synchronize traces. As discussed, the Convex-hull method can synchronize traces with better accuracy so it is chosen as the default option.

- **Reduction module**: the Reduction module consists of two methods. The first method is the previous work [1] where all node pairs are examined and no accuracy is possibly traded off for performance: “Accuracy method” [1]. The second method is our proposed method based on time reduction. We explain the proposed method in the next Section. To sum up, the previous method sends matched packets to the Analysis module without any processing and then synchronizes each node pair based on synchronization algorithm. Then, the reference time is specified and all nodes can be synchronized with this reference time in Reduction module. We therefore have to choose a node as reference node which has the most accurate links. Since each node pair in a computer cluster had to be synchronized in the previous method, there may
be indirect paths between two nodes with better accuracy (less drift and offset). To find which links (and drift/offset) to use for synchronizing two nodes, a Minimum Spanning tree (Min-ST) based on drifts and offsets is computed since it gives best accuracy to compare any two nodes in the cluster. The last part in this module is propagation. All nodes should be synchronized based on the reference node and drift and offset factors ($\alpha$ and $\beta$) propagated through the routed path to the reference node.

3.1. Main challenge

As we discussed, previous work tried to obtain the best accuracy for network synchronization [1]. In real setups, a scalability problem arises. The total synchronization time increases rapidly with respect to the network size. For example, in experiments generated with the NS2 network simulator, where there are 21 connected computers, the simulated time was seven seconds and during this period, each computer sent and received around 1000 to 4000 packets. The timestamp of each packet helps us to synchronize computers as we have seen before. The output of this simulation was sent to the previous accurate synchronization algorithm [1]. The total synchronization time for such an experiment, which is a realistic setup, is about 27 minutes. Obviously, it is impractical to wait 27 minutes to synchronize 21 computers that have been traced for just seven seconds. This scalability problem grows when adding more computers to trace and analyze simultaneously.

4. Methodology

To tackle the discussed challenges in Section 3.1, we ignore and neglect inaccurate links in the network. Let’s take a look at a simple example. Figure 2 shows four connected computers, A, B, C and D. In Figure 2.a, each number labelling an edge represents the number of exchanged packets in the link between that pair of nodes. In previous work, each pair of connected computers were synchronized. This meant computing the drift, offset and accuracy for each pair of connected nodes, A-B, A-D, A-C, B-C and D-C. The direct connection is considered and other indirect connections are considered as a combination of some direct connections. Figure 2.b displays the accuracy on each edge.

![Figure 2. Main graph with 4 nodes.](image)

Then, one of the nodes is selected to be the reference node in the cluster by the Reduction module and its time is assumed as reference time. Adding up the factors of the connected nodes gives total accuracy for each node. The node that has the smallest total bound on accuracy has the smallest error, normally associated with lower latency links to the other nodes [1]. Therefore, the synchronization between this node and others will be achieved with higher accuracy (here we have $A$: $1.10 + 0.12 + 0.34 = 1.56$). Thus, Accuracy method selects node “A” as reference node. All nodes in the cluster should be synchronized with the reference node. However, there are usually some nodes that do not have a direct connection to the reference node in the network. Furthermore, some nodes may have more than one indirect path to the reference node. Thus, a unique routed path from each node to the reference should be found. To this aim, a Degree-Constrained Spanning Tree [15] is sufficient. This tree connects all nodes together and shuns loop or redundant paths. In Accuracy method, the Reduction module finds Min-ST based on synchronization values [1]. In this example, ignored links are two links; A-B and D-C. A manifest disadvantage of the method is that it has synchronized these links already and now they are ignored and useless after this. The idea is “not to lose time finding these useless factors”. Let us now discuss the main features of our methodology:
4.1. Hop count

A criteria that leads us to find accurate paths is the number of hops. Since network delays cause a major part of errors, the accuracy degrades with the number of links from the source node to the destination node. The source node has hop count equal to zero and every node that has a direct link to it has hop count equal to one. Other nodes can count their hop count as the number of links from each one to the source node by incrementing the previous node's hop count with one.

This property is one of the operations used to select not only the reference node but also the routed path to propagate synchronization factors. We compose this criteria with another one, as explained in the next Section.

4.2. Number of exchanged packet

The tests results of the previous method and studies about factors impacting on the accuracy of connection synchronization show that the number of packets exchanged has a most significant impact on accuracy. Thus, we exploit this property to analyse links before synchronizing.

4.3. Reference node selection

Since a cluster includes many connected computers, the first step of optimization is to recognize the best criteria to choose the reference node. The reference node time is considered as reference time and drift and offset times are then calculated to other computers. We propose to select the reference node based on two criteria: 1) the fewest hop count to all nodes and 2) the maximum packet traffic. The first criterion is considered more effective and important. We select a set of nodes having the fewest hop count in the network. In the case where there is a single member in the set, we have found the reference node. Otherwise, we use the second criterion in order to have the minimum latency and delay. Thus, we select from the set the nodes with the highest packet count. Most of the time, these two conditions result in a single reference node candidate. In the case where more than one node qualifies, one is selected at random among these.

4.4. Optimized synchronization time

In the next step, after finding the reference node, we restrict the synchronization computations to the accurate paths to optimize the synchronization time. We obtain a Maximum Spanning Tree (Max-ST) based on the number of exchanged packets to all connected nodes, thus removing any redundant link. In this method, this Max-ST is sent to the Analyse module to synchronize the connected computers. Then, the synchronization factors must be propagated through the Max-ST to synchronize the time of all computers in the network cluster. Let's take a look at the previous example and go through these steps. In the proposed method, Max-ST for each link is used. The specific feature of this Max-ST is that it starts to grow with the Reference as root to the Max-ST. In the optimization step, as we discussed beforehand, the number of exchanged packets is counted and reference node (A) and Max-ST are selected. Figure 3 shows the Max/Min-ST that are exactly the same as with the previous method. The important difference between the two methods is the required computation time. Figure 1 illustrates the optimization variant in the Reduction module. With the proposed method, Max-ST is sent to the Analysis module to synchronize the nodes. The optimized method produces a significant reduction in computation time, while often producing the same or very similar synchronization accuracy results.

5. Real world and Experiment Results

5.1. Network Simulation

Since a large cluster is not always available for experiments, we used a powerful simulator to test more configurations and eventually validate our simulation results using Mammouth as a real world experiment. Many simulator, such as Wireshark, Clownix, and etc., were studied and finally NS2 was selected. NS2 provides substantial support for most network features on both wired and wireless types. It supports TCP, multi-cast, and routing protocols. NS2 simulates not only network latencies but also the distances and many other features.
The simulator has a clock and all node time sources are based on it. However, for synchronization studies, we need to run each computer node with its own independent clock, something none of the simulators offer. Therefore, the core of NS2 has been changed and an independent clock has been provided for each computer. Each computer sends and receives packets recording the events with its own clock and the results can be used as the input for synchronization algorithms.

Table 1. Timing improvement

| No. of Nodes | Total No. of Packets | Previous Sync. Time | Optimized Sync. Time | Saved Time | Percentage |
|--------------|----------------------|---------------------|----------------------|------------|------------|
| 4            | 1437                 | 8 s                 | 5.5 s                | 2.5 s      | 29%        |
| 5            | 2098                 | 13 s                | 7.5 s                | 5.5 s      | 41%        |
| 6            | 13044                | 80 s                | 69.5 s               | 10.5 s     | 13%        |
| 21           | 173985               | 1157 s              | 922 s                | 235 s      | 20%        |

The first network that we have simulated had four computers connected with a router. Each link has its own distance and delay. Also, nodes send packets with specified sizes at specified intervals through the links. Then, we expanded our network to 21 nodes and got the same accuracy as the Accuracy method in less time. Table 1 shows the timing improvement with the new method for our experiments.

The variation of the percentage mainly depends on the number of ignored links in the optimization step, the more packets are exchanged in the ignored links, the more time is saved. Figure 4 illustrates the network connections of the large experimental cluster with 21 nodes.
In this experiment, the number of packets exchanged between each pair of nodes varies from 500 to 4000, which is more realistic. The optimized method still achieves the highest desired accuracy while improving the total execution time. The result is shown in Table 2. This experiment has been repeated for many models and we obtained improvement for all of them.

![Figure 6. Comparison between optimized and previous methods](image)

Figure 5 illustrates the fact that the traffic of a network is variable. The experiments were performed in different sizes of networks and different intensity of traffic to measure their impact on performance. For example, there is a network with 16 nodes that has higher traffic compared to another network with 19 or 21 nodes.

Figure 6 illustrates the difference between two synchronization method compared for several networks. When there is a network with high traffic, synchronization takes more time. For example, in the network with 141123 packet exchanges, the synchronization methods take much more time than when there is less traffic.

![Figure 7. Saved time percentage](image)

Furthermore, it shows the time difference between the previous and optimized methods. As expected, the saved time depends not only on the network traffic but also on the network topology. The synchronization takes more time for a network with more nodes. For example, the total...
synchronization time for a network with 21 nodes, even with fewer packet exchanges (173985 packets) is higher than for a network with 16 nodes with more packets (209070 packets). Thus, the more nodes to analyze, the more time to synchronize the network. The saved time also depends on the features of the network. If a significant amount of network traffic consists in client to client communications (for example in distributed systems and high performance computing system), not only the number of ignored links is considerable but also ignored links sustain a high traffic. In that case, the saved time is particularly interesting (more than 50%). On the other hand, in typical networks with servers sharing data for numerous clients (the type of the experimented network), the optimized method could on average save 30% of the total synchronization time. Figure 7 shows the percentage of saved time for several networks. The variation of the percentage mainly depends on the number of ignored links in the optimization step, the more packet exchanges in the ignored links, the more time saved.

5.2. Real-world experiment results

One particularly interesting experiment was run on the Mammouth cluster with 55 connected computers. Mammouth is one of the largest Linux clusters in Canada, and is located at the Centre de Calcul Scientifique in Sherbrooke University, funded by the RQCHP. It is divided into two partitions: Serial and Parallel. Serial consists in Pentium 4 computers connected by an Ethernet Gigabit network. Parallel consists in Opteron computers linked by an infiniband network. The parallel partition includes 1120 Intel Xeon 64bit, 3.6 GHz CPUs in 560 Dell SC1425 compute nodes. Each node has 8 GB RAM (Total 4.5 TB) and 160 GB disk space. Infiniband network non-blocking (800 MB/sec) SDR Cisco-Topsin are used to connect nodes which run Linux CentOS 5 [7].

We experimented with many types of network topologies, however we describe here one of the most typical setups to support our claim. Figure 8 illustrates our experimental cluster with 55 nodes. This graph is one of the most popular topologies in networks. Usually, computers are divided into local area networks with nodes having connections to each other in the group. There is also a master node, or gateway, in the group that connects the group to other groups. In this example, different numbers of computers has been assumed in each group in the cluster. The first group consists of 19 connected computers, the second group consists of 21 nodes and third group consists of 15 nodes. A script has been run in each node to generate packet exchanges between each node and its two next neighbor nodes as well as to its master node.

| Trace | Drift  | Offset (*E+009) | Start time  | Trace | Drift  | Offset (*E+009) | Start time  |
|-------|--------|----------------|-------------|-------|--------|----------------|-------------|
| 0     | 1.000010 | 8.46           | 7605.594802 | 11    | 1.000000 | 8.15           | 7606.28     |
| 1     | 1.000000 | 8.92           | 7606.686668 | 12    | 1.000020 | 5.07           | 7606.285858 |
| 2     | 1.000080 | 3.57           | 7607.793202 | 13    | 0.999998 | 5.02           | 7606.291110 |
| 3     | 1.000040 | 4.67           | 7606.285003 | 14    | 1.000030 | 4.98           | 7606.289300 |
| 4     | 1.000030 | 4.86           | 7606.288089 | 15    | 1.000040 | 8.38           | 7606.294055 |
| 5     | 0.999996 | 1.45           | 7606.294936 | 16    | 1.000010 | 5.26           | 7606.293697 |
| 6     | 0.999998 | 5.06           | 7606.284456 | 17    | 1.000030 | 4.95           | 7606.284242 |
| 7     | 1.000020 | 8.81           | 7606.294419 | 18    | 1.000020 | 5.12           | 7606.292262 |
| 8     | 1.000030 | 4.97           | 7606.292909 | 19    | 1.000000 | 4.83           | 7606.285830 |
| 9     | 1.000030 | 5.65           | 7606.290214 | 20    | 1.000030 | 8.56           | 7606.28     |
| 10    | 1.000030 | 4.80           | 7606.285457 |        |        |                |             |
It is obvious that there are more connections to the master nodes. In our experiments, each connection is a "ssh connection" and generates from 43 to 48 packets. The previous method finds the best links, with the best time synchronization accuracy, then synchronization factors are propagated through these links and all nodes are synchronized based on the reference node’s time. Figure 9 displays the final accurate graph in the experimental cluster. Node 1 is selected as the reference node and all 20 nodes of the second group can be synchronized directly with it. Also, each of the other group members can be connected to the reference node through an indirect path through their own master node. Both methods synchronize the nodes through this accurate graph. However, the new proposed method is able to reduce the computation time. Performance improvement depends on the number of ignored links. Thus experiments with real connections and numerous messages are useful to compare the performances of these methods.

6. Conclusion and future work

In this paper, we have discussed offline trace synchronization for computer clusters with large number of communications. We have proposed a method to optimize the synchronization performance. The presented method is applicable when event data is distributed over a computer cluster, and exchanged packets are used for time synchronization. It has been integrated in the Linux Trace Toolkit next generation (LTTng), developed at École Polytechnique de Montréal, and used both for performance debugging and security and performance monitoring.

As future work, we plan to extend the method to synchronize online traces while receiving streaming data from LTTng. Since, in a streaming trace, nodes are discovered as they become active, and may cease operation or activity at any time, nodes joining and leaving at any time during the streaming trace cause alterations to the network topology and consequently complexify the analysis.

7. References

[1] B. Poirier, R. Roy and M. Dagenais, “Accurate Offline Synchronization of Distributed Traces Using Kernel-level Events,” Operating Systems Review, vol. 44, 2010, pp. 75-87.
[2] M. Desnoyers, “Low-Impact Operating System Tracing,” PhD thesis, École Polytechnique de Montréal, 2009.
[3] A. Duda, G. Harrus, Y. Haddad and G. Bernard, “Estimation global time in distributed system,” In proceeding 7th Int. Conf. on Distributed Computing Systems, pp. 299-306, Berlin, 1987.
[4] H. Khlifi and J. C. Gregorie, "Low-complexity offline and online clock skew estimation and removal," The International Journal of Computer and Telecommunications Networking, vol. 50, no. 11, pp. 1872-1884, 2006.
[5] E. Clement and M. Dagenais, “Trace synchronization in distributed networks,” Journal of computer system, Network, and Communication, 2009.
[6] A. D. Ksehmkalyani and M. Singhal, “Logical time,” in Distributed Computing: Principles, Algorithms, and Systems, 1st ed., USA: Cambridge University Press, 2008, pp. 50-84.
[7] Mammouth project available at “https://rqchp.ca/?mod=cms&pgid=566&lang=EN,” Sep. 2010.
[8] P. Ashton, “Algorithms for off-line clock synchronization,” Technical report, University of Canterbury, Department of Computer Science, Dec. 1995.
[9] R. Sirdey and F. Maurice, “A linear programming approach to highly precise clock synchronization over a packet network,” 4OR: A Quarterly Journal of Operations Research, vol. 6, no. 4, 2008, pp. 393-401.

[10] B. Scheuermann, W. Kiess, M. Roos, F. Jarre and M. Mauve, “On the time synchronization of distributed log files in networks with local broadcast media,” IEEE/ACM Transactions on Networking, vol. 17 no. 2, 2009, pp. 431-444.

[11] R. Sirdey and F. Maurice, “A linear programming approach to highly precise clock synchronization over a packet network,” 4OR: A Quarterly Journal of Operations Research, 6(4):393–401, 2008.

[12] H. Marouani and M.R. Dagenais, “Internal Clock Drift Estimation in Computer Clusters,” Journal of Computer Systems, Networks, and Communications, vol. 2008, no. 1, 2008, pp. 1-7.

[13] E. Betti, M. Cesati, R Gioiosa and F. Piermaria, “A global operating system for HPC clusters,” IEEE International Conference on Cluster Computing and Workshops, 2009.

[14] NIST Time and frequency from A to Z., February 2011. http://tf.nist.gov/general/glossary.htm.

[15] Raidl, G.R., Julstrom, B.A., "Edge sets: an effective evolutionary coding of spanning trees," Evolutionary Computation, IEEE Transactions on, On page(s): 225 - 239, Volume: 7 Issue: 3, June 2003

[16] J. H. Deschenes, M. Desnoyers and M. Dagenais. “Tracing Time Operating System State Determination,” The Open Software Engineering Journal, vol. 2, 2008, pp. 40-44.