

# **DLB – A Dynamic Load Balancing Tool for Grid Computing**

R.U. Payli, E. Yilmaz, A. Ecer, H.U. Akay, and S. Chien

Computational Fluid Dynamics Laboratory  
Purdue School of Engineering and Technology  
Indiana University-Purdue University Indianapolis  
Indianapolis, Indiana 46202 USA  
*Email: eyilmaz@iupui.edu, Web: <http://www.engr.iupui.edu/cfdlab>*

Keywords: load balancing, grid computing, parallel algorithms, distributed computing

## **1. INTRODUCTION**

The current computational power demands and constraints of organizations have led to a new type of collaborative computing environment called Grid Computing [1]. Grid Computing is a new way of the parallel and distributed computing. With grid computing, an individual can unite pools of servers, storage systems and networks into a large system. End users and applications see this environment as a big virtual computing system. The systems connected together by a grid might be in the same room or distributed globally, running on multiple hardware platforms, running different operating systems, and owned by different organizations. A cluster, a network attached storage device, a scientific instrument, or a network is not a grid by itself, but each might be an important part of the grid. While simultaneous resource allocations can be done manually with privileged accesses to the resources, such environments need certain resource management strategy and tools that can provide security and coordination of the resource allocations.

There are different approaches in management of loads in parallel clusters. Most accepted approach is using job schedulers to manage loads in the clusters based on rules associated with job types, load classes, etc. This is a system-level centralized management strategy which works over all users and jobs. The other approach is user-level individual management of loads in parallel computing environment. In most CFD applications, grids are migrated from one parallel task to another to balance the loads on computers. However, this method assumes that equal distributions of parallel task size is best assuming that loads in parallel clusters are homogeneous or they are using dedicated systems. Another approach which is followed in this study is in between these two approaches such that one can balance loads by actually moving parallel tasks among available parallel compute nodes. This gives more rooms to user-level load balancing and eventually let the system come to balance without going through centralized load management or data migration among parallel tasks.

In this paper, we present a version of Dynamic Load Balancing, DLB, which has been developed at the IUPUI CFD Laboratory during last ten years [1-5]. Some changes have been

made to the previous version to make it more compact and distributable with a three dimensional test case such that those who are interested in using DLB can have a complete package in a compact form. We demonstrated applicability of the environment by using a parallel example program for CFD applications. Next section is devoted to description of the DLB and its components. The differences of this version from previous version are also discussed. This follows with 3D parallel test code to be used with DLB for testing purposes. Then, implementation of DLB into a CFD code is explained. Finally, results of DLB with this test case are introduced with final conclusions drawn.

## 2. DYNAMIC LOAD BALANCING

DLB is used to provide application level load balancing for individual parallel jobs. It ensures that all loads submitted through the DLB environment are distributed in such a way that the overall load in the system is balanced and application programs get maximum benefit from available resources. Current version of the DLB has two major parts. One is called System Agent that collects system related information such as load of the system and the communication latency between computers. The other is called DLB Agent which is responsible to perform the load balancing. System Agent has to run all configured machine on the environment whereas DLB Agent is started by the user. The structure and components of the DLB environment is shown in Figure 1. Major components of the DLB are System Agent, and DLB Agent. Both components are written with Java. System requirements for DLB are LINUX/UNIX Operating Systems, Java 1.4 for System Agent (recommended) and DLB Agent and parallel environment supported by MPI implementation. Implementation and user manual of DLB is given in [6].

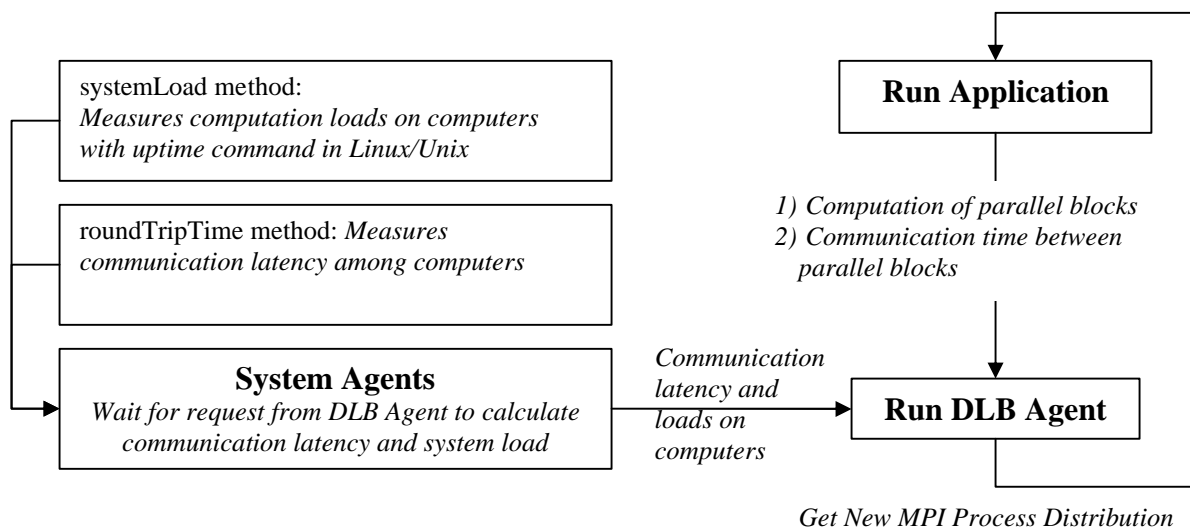


Figure 1: Components of DLB.

### 2.1. System Agent

System Agent acts as a server. A copy of System Agent has to run on all machines where users want to run their applications. System Agent provides system load information about the machine which is running through *systemLoad* method. Also provides communication timing between two hosts through *roundTripTime* method. *systemLoad* method issues Linux/Unix *uptime* command and gets the last 15 minutes system load average which is given

by the *uptime* command to determine the system load. *Uptime* represents load average which is a rough measure of CPU usage. It reports the average number of processes active during the last minute, the last five minutes, and the last 15 minutes. High load averages usually mean that the system is being used heavily and the response time is correspondingly slow. *roundTripTime* method calculates the round trip time of a 1024 bytes message sending and receiving between source and destination hosts.

## 2.2. DLB Agent

DLB Agent collects system load information from system Agents which are running all the hosts selected by users. It measures communication timing between pairs of the hosts and do the load balancing based on these information. DLB Agent has 4 methods to do above tasks. These methods are *measureCommunicationLatency*, *getCommunicationInfo*, *getSystemLoadInfo* and *loadBalancer*. *measureCommunicationLatency* method contacts System Agent on the each host which are in the host information file of the user and calculates the communication time between each pair of hosts using the System Agent's *roundTripTime* method. *getCommunicationInfo* method retrieves the communication time information for each pair of hosts which is calculated by the *measureCommunicationLatency* method and generates the communication information file. *getSystemLoad* methods contacts all the System Agents which are in the user's host lists and gets the system load information which is calculated by the System Agent's *systemLoadInfo* method and generate the system load information file. *loadBalancer* method does the load balancing based on the load and communication timing information of the machines, the application's timing and interface information. *loadBalancer* moves to a process (block) from the loaded machine to the least loaded machine to reduce total elapsed time.

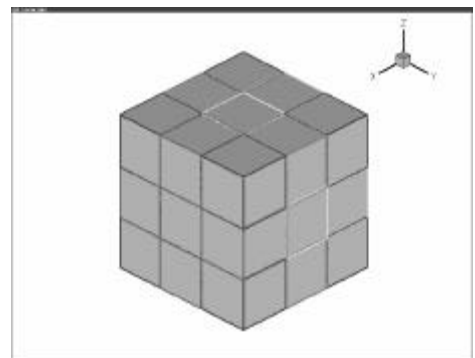
## 2.3. Comparison with Previous DLB Version

Previous System Agent Implementation	New System Agent Implementation
<p><u>System Load:</u> System Agent gets the system load information from <i>PTrack</i> – a standalone program which runs on all the machines. <i>PTrack</i> counts processes running on the system by using <i>ps</i> command in Linux/Unix. It runs continuously on the system and records average loads other than system related loads and processes not submitted through the DLB.</p> <p><u>Communication Latency:</u> System Agent gets the communication latency between each pair of hosts from <i>Crack</i> – a standalone program which runs on all the machines. <i>Crack</i> sends 1024 bytes of the data between source and destination computers, then measures round trip time for this process. <i>Crack</i> is considering all IP addresses (0-255) in the subnet whether System Agent exists there or not. If it is there, it gets communication latency between computers by the method mentioned above. Drawback of this method is excessive time to check all IP addresses, even if they do not assign any computer for some IP addresses.</p>	<p><u>System Load:</u> System Agent runs a Linux/Unix command <i>uptime</i> to get load information on each computer, whenever a request made to get system load by DLB Agent. It does not run continuously on the system but it rather gets load information on-demand base only.</p> <p><u>Communication Latency:</u> To get the communication latency between computers, a function in System Agent called <i>roundTripTime</i> is used. This function sends 1024 bytes of the data between source and destination computers, then measures round trip time for this process. It is same as previous versions of DLB. The advantage of this implementation is that it only considers host computer list of the users to save system overhead time.</p>

Previous DLB Agent Implementation	New DLB Agent Implementation
<p>First DLB Agent checks existence of the <i>procdef</i> file, which contains process distribution regarding parallel blocks. If this file does not exist, it generates initial <i>procdef</i> file by using the computer load and relative computer speed. It checks the host on which application program has to start. It starts Job Agents to control the input and output files.</p> <p>Then it starts <i>InterClusterCtrack</i> to calculate communication cost between two different subnets. Starts the application program with user supplied script. Waits for timing information file from the application program. When the timing information file is ready executes load balancer program. Load balancer program does the actual load balancing and generates a new <i>procdef</i> file. If the new distribution is better than the current one it generates new <i>procdef</i> file and the file to inform application program to stop. After application program stops DLB Agent starts the application program with new <i>procdef</i> file and wait for the new timing file.</p> <p>It is hard to start and stop a parallel application. It is not always the case cleanly stop the application program. When the application program is crashed user has to manually clean the left over processes from all the machines.</p>	<p>DLB Agent prepares system load information file and communication time file and does the load balance. DLB Agent has 4 methods to do these tasks. These methods are:</p> <p><i>measureCommunicationLatency</i> method contacts System Agents on the each host which are in the host information file of the user and calculates the communication time between each pair of hosts using the System Agent's <i>roundTripTime</i> method.</p> <p><i>getCommunicationInfo</i> method retrieves the communication time info for each pair of hosts which is calculated by the <i>measureCommunicationLatency</i> method and generates the communication info file.</p> <p><i>getSystemLoad</i> methods contacts all the System Agents which are in the user's host lists and gets the system load information which is calculated by the System Agent's <i>systemLoadInfo</i> method and generate the system load information file.</p> <p><i>loadBalancer</i> does the actual load balancing. If the new distribution better than the old one it generates new <i>procdef</i> files for MPICH and LAM/MPI and a file which indicates that the new distribution is better than old one.</p>

### 3. A PARALLEL CFD TEST CASE CODE FOR DLB

A 3D transient heat equation is implemented on structured grids with forward time central space (FTCS) differentiation method [7]. Here the grid partitioning is done in three directions whereas in the previous version we were partitioning in one direction only. Therefore, maximum neighbor blocks were two in the previous cases. But now in three directional partitioning, there is 6 blocks connected to each other maximum. Therefore, communication times increase hence wait time for each process. Figure 2 shows the 3D grid used for heat transfer problem. Boundary conditions are 0 and 1 in lower and upper surfaces respectively and linearly changing in all other surfaces. Our emphasis is on solving real CFD



**Figure 2: 3D Grid boundary of 27 blocks for parallel CFD test case.**

problems while reducing complexity due to boundary conditions as it would be different for different problems. Figure 2 shows 27-block partitions of 3x3x3 in all directions that each has 40x40x40 grid points, which ends up with 1.728 millions of grid points.

#### 4. IMPLEMENTATION OF DLB IN CFD CODE

As far as user interaction goes, DLB requires two files for new block calculation: one is timing of computation for parallel blocks, and second one is interface size of each block with its neighbors. Timing file, *timing.in*, contains block ID, block grid size, and elapsed computation time excluding communication time. Interface file, *iconnect.in* contains, block ID, neighbor ID and interface grid size. Following is a simplified coding of the main program for the application used here. It includes only timing data extraction part from the solver main part. Interface file can be generated at the grid reading part of the program as it will be read directly from partitioned grid data file, which is prepared by grid generator program. MPI is the parallelization library used for this program; therefore timing function provided by MPI is used to get elapsed time for block. Note that communication time should not be included in block time. Therefore it is subtracted at the end from the total time. Italic parts are timing related implementation to the solver and bold corresponds to solver implementation.

```

Program PCFD_Solver
! initialize parallel environment
  call mpi_initialize()
  myid = mpi_comm_rank(...,myid,...)
! read grids
  call gridRead()
! get start time for block
  etime1blk = mpi_wtime()
! time stepping
  Do iTime=1, nTime
    call boundary(...)
    call heatSolver(...)
! get communication elapsed time
    etime1int = mpi_wtime()
    call interface_communicate(...)
    etime2int = mpi_wtime()
    etimeint = etimeint + (etime2int-etime1int)
  EndDo
! get final time and total elapsed time for block
  etime2blk = mpi_wtime()
  etimeblk = etime2blk-etime2blk
  etimeblk = etimeblk - etimeint
! write time information into timing file
  write(2,*) myid, nelem, etimeblk
  call mpi_finalize()
End

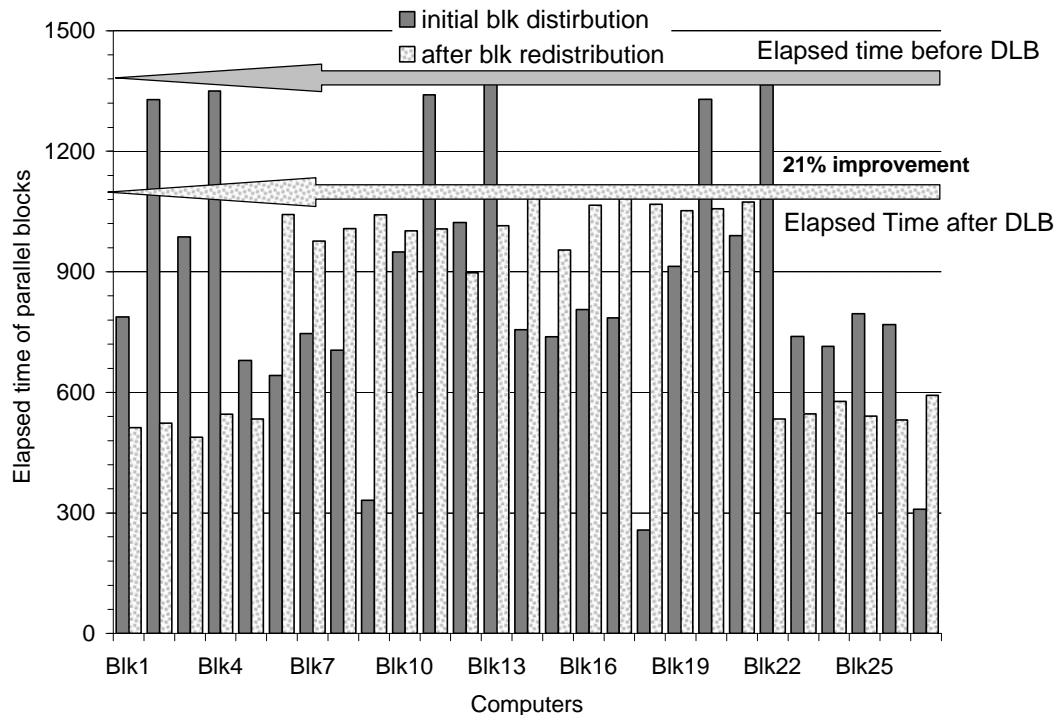
```

## 5. DLB RESULTS

Parallel CFD test case has been run with the grid given in Figure 2. Computers of heterogeneous operating systems, AIX, Linux, and Solaris, are used. A computer cluster composed of 4 IBM RS6000, 4 PIII Linux, and one 2-CPU Sun Ultra computers is used at CFD laboratory in IUPUI. As further improvements are still ongoing to add user-interactive parts, in this study, only the advantages of DLB has been shown with new functions to measure computer loads. Table 1 shows parallel block distribution before and after DLB was used for block redistributions. Initially, an even block distribution was chosen, then DLB redistributed accordingly by considering loads on computers and communication among processors.

**Table 1: Parallel block distribution and total system loads before and after DLB.**

Hostname	aix1	aix2	aix3	aix3	linux1	linux2	linux3	linux4	solaris
Parallel Block Distribution									
<i>Cycle 0: Initial blk distribution</i>	3	3	3	3	3	3	3	3	3
<i>Cycle 1: After blk redistribution</i>	2	1	2	0	4	4	4	4	6
Total System Loads									
<i>Cycle 0: Initial blk distribution</i>	2.47	3.76	2.62	3.88	1.2	1.11	1.5	1.19	0.64
<i>Cycle 1: After blk redistribution</i>	1.97	2.38	2.04	2.05	3.02	2.87	3.14	3.05	2.71
<i>Relative Speeds of Computers</i>	1.02	1.00	1.01	1.00	1.19	1.19	1.20	1.17	1.55



**Figure 3: Parallel blocks elapsed time before and after DLB.**

Relative speeds of the computers based on CPU time with same compiler in all machines are 1.00 for Aix, IBM RS 6000, 1.18 for Linux, PIII, and 1.54 for Solaris, Sun Ultra. Blocks are moved to faster and available computers, which are in this case Solaris and Linux machines. After blocks are moved loads on the systems are changed accordingly. Figure 3 shows elapsed time change of parallel blocks with initial distribution and after DLB is used. It is obvious that DLB provides improvements in elapsed timing of the parallel jobs. Elapsed time improvement is about 21 percent from initial distribution.

## 6. CONCLUSION

We have demonstrated a compact form of DLB integrated with a parallel CFD test case program applicable to Linux and Unix flavored computers in Grid environment. System load measurement of DLB is modified with average load history provided by Unix/Linux systems rather than tracking processes by system agents of the DLB package. In addition, load balancer program has been implemented in Java same as all other units of the DLB to make it all in one language. Parallel simulations have shown that DLB makes significant improvement for better load balance on the system. File I/O and running DLB is smooth. Using LAM/MPI makes program start and halt easier than MPICH due to starting/halting parallel environment with daemons running on all compute nodes. In this version, user provides script file to start/halt all programs and environments and running of DLB for several cycles. This can be automated further by providing a better user interface to DLB package.

## 7. ACKNOWLEDGEMENTS

The authors greatly appreciate the financial support provided by the NASA Glenn Research Center, under Grant No. NAG3-2260.

## 8. REFERENCES

- [1] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today*, 55 (2) 2002, pp. 42-47.
- [2] Y.P. Chien, A. Ecer, H.U. Akay, F. Carpenter, and R.A. Blech, "Dynamic Load Balancing on a Network of Workstations for Solving Computational Fluid Dynamics Problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 119, 1994, pp. 17-33.
- [3] Y.P. Chien, J.D. Chen, A. Ecer, H.U. Akay, and J. Zhou, "DLB 2.0 – A Distributed Environment Tool for Supporting Balanced Execution of Multiple Parallel Jobs on Networked Computers," *Proceedings of Parallel Computational Fluid Dynamics 2001*, Egmond aan Zee, The Netherlands (May 21-23, 2001), Elsevier Science B.V., Amsterdam, The Netherlands, 2002, pp. 95-102.
- [4] E. Yilmaz, A. Ecer, H.U. Akay, R.U. Payli, S. Chien, and Y. Wang, "Parallel Computing in Grid Environment," *Proceedings of Parallel CFD 2003*, (May 12-15, 2003), Moscow, Russia.
- [5] S. Chien, Y. Wang, A. Ecer, and H.U. Akay, "Grid Scheduler with Dynamic Load Balancing for Parallel CFD," *Proceedings of Parallel CFD 2003*, (May 12-15, 2003), Moscow, Russia.

- [6] R. Payli and E. Yilmaz, “DLB for Grid Computing: Implementation and Manual”, Technical Report, CFDL 04 01, Mechanical Engineering, IUPUI, Indianapolis, IN, USA, 2004.
- [7] V. Kargin and E. Yilmaz, “A Parallel CFD Test Case: Solution of Heat Equation”, Technical Report, CFDL 04 02, Mechanical Engineering, IUPUI, Indianapolis, IN, USA, 2004.