# A Generalized Environment for Distributed Image Processing

Haresh S. Bhatt * and A. K. Aggarwal §

* Computer and Network Facility
Signal Image and Information Processing Area
Space Applications Centre
Ahmedabad , INDIA
Email: haresh@ipdpg.gov.in

§ School of Computer Science
University of Windsor
Windsor, CANADA
Email: akshaia@uwindsor.ca

## Abstract

Image processing applications (IPA) requirements can be best met by using the distributed environment. Development of an application having in-built automated data transfer, capability of using multiple machines in parallel, and robust error handling is a challenging job.

This paper presents the design of a Development Environment for Distributed Image Processing (DEDIP) based on Master-Slave model that provides a user-friendly image processing operational environment.

The DEDIP has been tested for five cases using three simulated packages by 3 operators in 10 runs. Furthermore, fifteen scientists have used the environment to develop ten image-processing applications for distributed processing. The system is now operationally being used for the ten applications.

DEDIP provided a much higher efficiency (223% for parallel application to 106% for sequential application) in comparison with the task carried out manually. Furthermore, DEDIP gives 90-95% efficiency compared to the best theoretical expected one. This is an excellent figure particularly because the application designer achieves it with very little effort, for development of the distributed application.

**Key Words:** DEDIP, image processing, parallel image processing, distributed image processing, ease of use

## 1. Introduction

The research and development work in parallel & distributed-processing area concentrates on efficient usage of the technology, not the ease of use [1-17]. Most of the work aims at supporting the higher computation requirement of large applications. The image processing applications are not that large. However, they need higher processing power due to large volume of data sets (images). Many small different applications are executed on different data sets operationally. These applications may require different resources too. Hence an optimum usage of resources is needed.

The research work carried out by eminent computer professionals [1-17] focused only on the parallel-programmers' needs. The image processing applications are developed by scientists (mathematicians, physicists, remote sensing experts, etc).

They are not expert in parallel programming. We have developed a full-fledged Development Environment for Distributed Image Processing (DEDIP) that makes the development & operationalization of distributed applications very easy.

## 2. Factors for the Design of the Generalized Environment

The data product generation scheduler (DPSCH) support [18] was an attempt in providing smooth operational environment for IRS (Indian Remote sensing Satellites) data product generation. It was developed using the job shop-scheduling model [19]. CVS Prakash has shown the usage of the application scheduler to share a special purpose resource among the Unix based image processing & analysis systems [20]. However DPSCH is an application. The author decided to design an integrated environment, which may be user-friendly.

The authors designed a model, DEDIP (Development Environment of Distributed Image Processing), based on middle-ware architecture extending & generalizing the work of [18] and [20] in multiple dimensions to cater to all the above requirements.

DEDIP supports multiple process scheduling. The process names and resources can be configured, added and altered any time. The DPSCH schedules the processes in a fixed sequential manner. The DEDIP design generalizes it to schedule processes as per the user's choice. The user can configure it in sequential, parallel or hybrid scheduling pattern. The DEDIP reads the process scheduling pattern from the user's configuration file. The configuration file can be altered as and when required before initiating the application.

DEDIP supports a network of multiple heterogeneous systems. The user has to simply provide the required information in his configuration file. DEDIP supports automated file transfer among the nodes as required by the applications' components. The data transfer requirement may be static for some applications and dynamic for other applications. The user can manually provide the static information in the predefined file once for all. DEDIP supports a library routine for generating this information dynamically during the application execution. The user has to simply call the routine with the required information. The routine will generate the required file in a predefined format. DEDIP supports multiple simultaneous applications. It will not only execute the tasks configured by each application but also provides various status displays to the operator to monitor and control his session.

The user configures the synchronization point for his application. DEDIP carries out the required synchronization using its resource management library, for all the resources required by an application.

DEDIP supports error handling for abnormal termination of any process. In addition DEDIP extends the same to support the application level implications. If any error occurred in any task of an application, DEDIP provides the facility to restart the task, restart the entire application, suspend/resume the application or abort the application. It handles all the tasks (on host and on remote nodes) accordingly. DEDIP can schedule the interactive processes on X11 terminals.

DEDIP supports GUI (Graphical User Interface) on all the platforms. This version of DEDIP started with enhancing the interaction capabilities on VAX/VMS. Then the user friendly GUI was developed on Unix, using Motif. Lastly was developed the Java based GUI.

The DEDIP has automated the application building operations if user provides his source code and makefiles. Incase the user does not want to give the source code; he will have to build the application components on all the nodes and copy the executable at predefined locations. The DEDIP uses a simple meaningful directory structure that is easy to remember.

**3. What user needs to do:**

The DEDIP users are application designers, operation managers and operators. The application designer has to simply break his large application into small processes, which constitute his application. The break-up depends on resource requirement as well as processing requirements. Then the designer has to create & use the intermediate files to integrate the processes. Most of the scientists are very much familiar with this mode of working. The designer of the application should create makefiles for each process. He can test his entire application on a single node. He should prepare an error-list containing error codes, error messages and possible recovery actions.

Then he configures the following information: (1) process interdependency information; (2) resource requirement of each process; (3) nodes on which the process can be executed; (4) list of intermediate files in static dependency; (5) makefiles; and (6) error-list.

The operation manager can modify the configuration file whenever needed. He may do it whenever any system is added or removed. He can also prepare multiple configuration files: e.g. (1) for off-hours to use the personal computers; and (2) for regular usage.

Either application designer or the operation manager can build the application using DEDIP.

The operator has to select the application using the user-friendly navigation and execute it. He can execute multiple applications simultaneously. He can execute one application multiple times for different data sets. He can monitor and control his session through one integrated GUI for all applications.

**4. DEDIP overview:**

The DEDIP is based on the master-slave model. The model can be visualized from figure-1. As seen from figure-1, the master consists of three processes running on the host, OPRINT, HostManager and Server while the slave consists of two processes running on workstation, Server and WSManager. DEDIP environment supports multiple hosts and workstations.

The OPRINT is a graphical user interface, which interacts with users. It interacts with the application designer or operation manager to build the application. It displays the complete information about the build status of each process on each node.

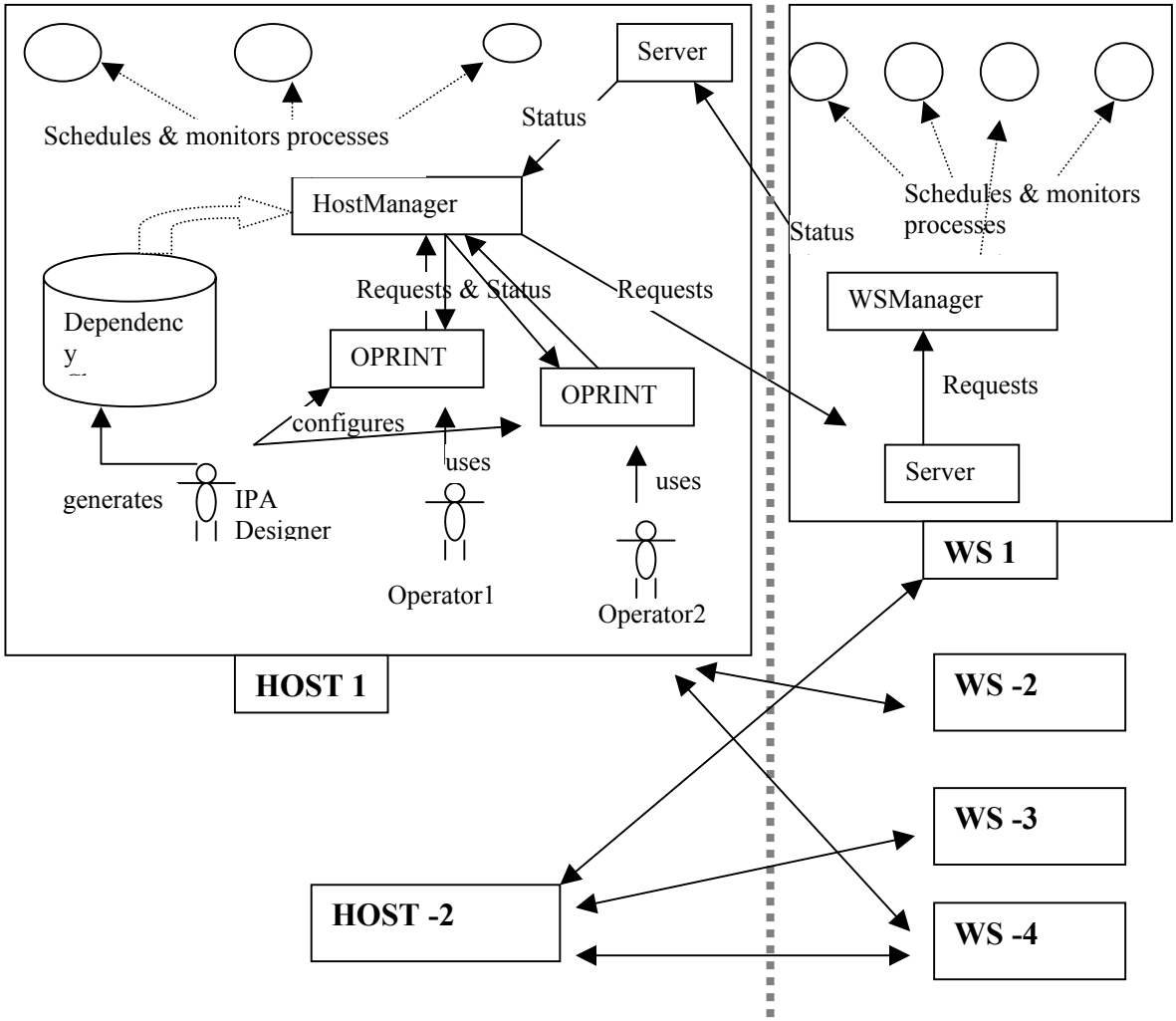



**Figure-1: DEDIP master-slave model**

The OPRINT interacts with the operator for following actions: (1) Application selection and execution; (2) Application monitoring and controlling; and (3) Error handling. The operator can filter the views for application overview and details about all processes of an application. He has following controls over the application; (1) he can suspend an application; (2) he can resume a suspended application; (3) he can abort an application; and (4) he can restart an application.

The OPRINT displays the error code, error message and recovery action when any process terminates abnormally. The operator can either restart the process/application or can abort the application.

The operator has following controls over his session: (1) *Immediate termination:* All the running processes of his session are terminated immediately; and (2) *Process wide Termination:* The session is terminated after successful completion of all the running processes. The DEDIP provides him the facility to continue with the session at the

next logon. Incase of the first option, the time spent by the running processes is lost while the second one does not have any such loss.

The OPRINT submits all the requests to the HostManager. The HostManager satisfies each request with the help of WSManager.

For example, OPRINT sends a request to the HostManager to start an application. The HostManager reads the application configuration file and generates a linklist. The HostManager gets the information about the first process from the linked list. It finds out the target node of the process. If the target node is the host itself then it schedules the process and monitors, else it sends the request to the Server process on the target node. The Server process passes the request to the WSManager. The WSManager schedules the process and monitors. On completion of the process, the WSManager sends the status information to the Server running on the host. The Server passes the status to the HostManager. The HostManager checks the status. For erroneous status, it informs OPRINT to interact with the operator for necessary actions. On successful completion of any process, HostManager finds out the dependent processes from the IPA link list and starts each of them. The HostManager transfers the files generated by the process to the target node before starting the dependent processes. The HostManager keeps track of each application & each process through out the session. The OPRINT provides the various displays to keep the operator informed about all the minute details about the application and the progress of the process.

The Server on host and workstation help HostManager and WSManager to communicate with each other. It takes care of multiple simultaneous requests.

The DEDIP carries out the optimum resource management using the callable resource management library in conjunction with the details provided by the user in a configuration file. This library was developed to support the resource allocation, release, availability, and file transfer facilities. It allocates the required resource before task scheduling. The resource allocation routine put the tasks in wait mode incase the resource is occupied by another task. It releases the resource after completion. It was observed during the operation of some lengthy tasks that it needs the resource only for a short duration. In such cases, the application designer can optimize the resource utilization using the library. The users are familiar with such programming as they have been allocating and releasing the resources in the manual mode. All the scheduler processes reside in hibernation (sleep) mode releasing all the resources to the scheduled processes. A process wakes up by an interrupt or an event. Thus the resource usage by the DEDIP is negligible.

The DEDIP keeps track of the entire session. Hence, it can restore the session with minimum loss of processing power in case of power failure.

## 5. Case study

The DEDIP was initially tested using five simulated test cases. Later it has proven its capability for distributed image processing application development and operationalization at National Remote Sensing Agency (NRSA). NRSA is one of the operational centers for IRS (Indian Remote sensing Satellites) image processing.

**5.1 Simulated test cases:**

The efficiency of the DEDIP can be measured in terms of (a) reduction in operator interaction; (b) reduction in the time delay due to operator intervention. (The time delay may be due to operator inefficiency, operator non-availability at the required time, and so on); and (c) elapsed time. The time information was measured inside each simulated process. Furthermore the scheduler also provides the same details for each process.

Five cases were simulated for evaluating the above parameters. The simulated processes were generated resembling the actual processes but for the image processing interaction/processing. The process dependency chart is given in Figures 2.a, 2.b, and 2.c. The elapsed time requirement and processing node is shown in the bracket. The DTHS stands for the Data Transfer from Host to Slave where as DTSH stands for the reverse process. The data volume and the expected transfer time are shown in bracket. 'T' indicates the tape unit requirement by the process. 'I' indicates that the process is I/O bound job.
In the absence of DEDIP environment, the operator will have to execute all commands manually or through command procedure and keep track of the operation.

*Case 1:* Single package requiring sequential scheduling is shown in figure 2.a depicting the simplest case. It requires operators' concentration.

*Case 2:* Single package requiring parallel scheduling is shown in figure 2.b. This is similar to that of case 1 but operator will have to decide the scope of parallel processing.

*Case 3:* Two packages execution, each package requiring sequential scheduling, is shown figure 2.c. The operator can execute both packages one after another from a single terminal, which makes the operation easy but decreases the throughput. The operator can execute both packages simultaneously from two terminals, which improves the throughput but complicates the operation. This requires a high degree of operator concentration and increases the probability of operator mistakes.

*Case 4,5:* Similar to case 2 and 3 but operator uses two terminals under non-DEDIP environment to achieve better throughput.

**Figure 2.a: Single Package**

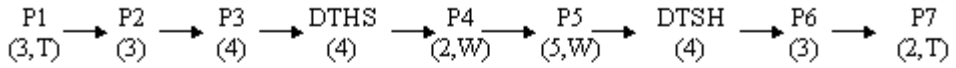



**Figure 2.b: Single Package having parallel processing**

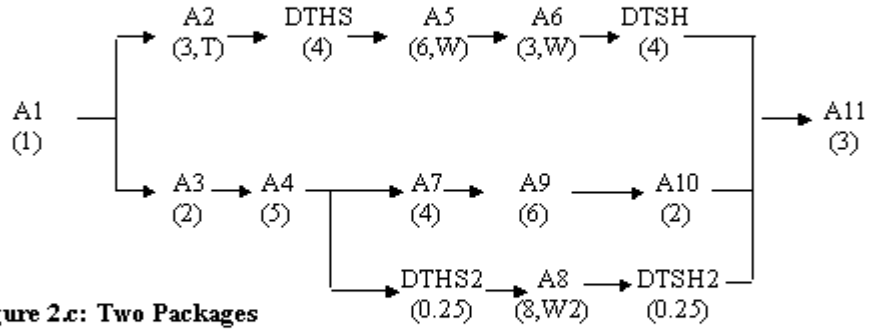
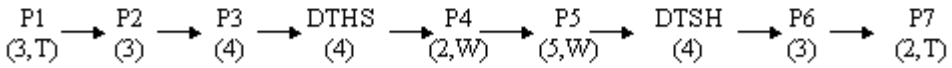



**Figure 2.c: Two Packages**

***Package 1:***




***Package 2:***

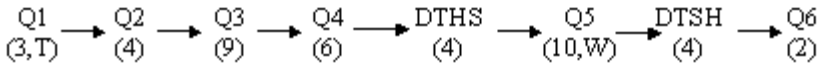



**Table 1: Results for the case studies**

| | Parameters | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Without DEDIP | | | | Using DEDIP | | | |
| **Case** | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** |
| 1 | M | 5% | 32 | 1 | VL | NG | 30.5 | 1 |
| 2 | H | 10% | 52 | 1 | VL | NG | 23.5 | 1 |
| 3 | H | 10% | 74 | 1 | VL | NG | 45.5 | 1 |
| 4 | H | 20% | 28 | 2 | VL | NG | 23.5 | 1 |
| 5 | VH | 20% | 49 | 2 | VL | NG | 45.5 | 1 |

M: Medium VH: Very High H: High
VL: Very Low NG: Negligible

A) Operator interaction
B) Time delay due to operator interaction and error recovery
C) Elapsed time (min)
D) Terminals used

The simulated applications were tested using system configuration consisting of VAX/VMS and Unix operating systems.

The results obtained by three operators in 10 runs, effectively 30 runs, for the test cases are listed in table 1. It is clear from the results that the DEDIP provides better throughput in case of the parallel-processing requirement. The time delay is a function of operator interaction, network data transfer, and optimal usage of the resources. The figures given are subjective to the operator efficiency and experience. It is seen that operator interaction and time delay are higher in non-DEDIP environment compared to DEDIP environment. They are much higher in the cases exploring the parallel resource usage under non-DEDIP environment to achieve better throughput.

DEDIP provided a much higher efficiency (223% for parallel application to 106% for sequential application) in comparison with the task carried out manually. Furthermore, DEDIP gives 90-95% efficiency compared to the best theoretical expected one (calculated from figure-2a and 2b). This is an excellent figure as it saves the valuable development efforts required by each application designer.

### 5.2 IRS-1C operationalization

The DEDIP was provided to the IRS-1C image process application designers and operational center for evaluation. DEDIP is used for development and operationalization of 10 IRS-1C packages (applications). The list of the packages operationalized is given in table 2. The system configuration is shown in figure 3. The High Density Digital Tape Recorder (HDDTR) is capable of storing large volume of data. It can be used to record the data in real-time too. The FSDU (Frame Sync and Decom Unit) is the special purpose H/W developed to record on HDTTR and read from HDDTR.

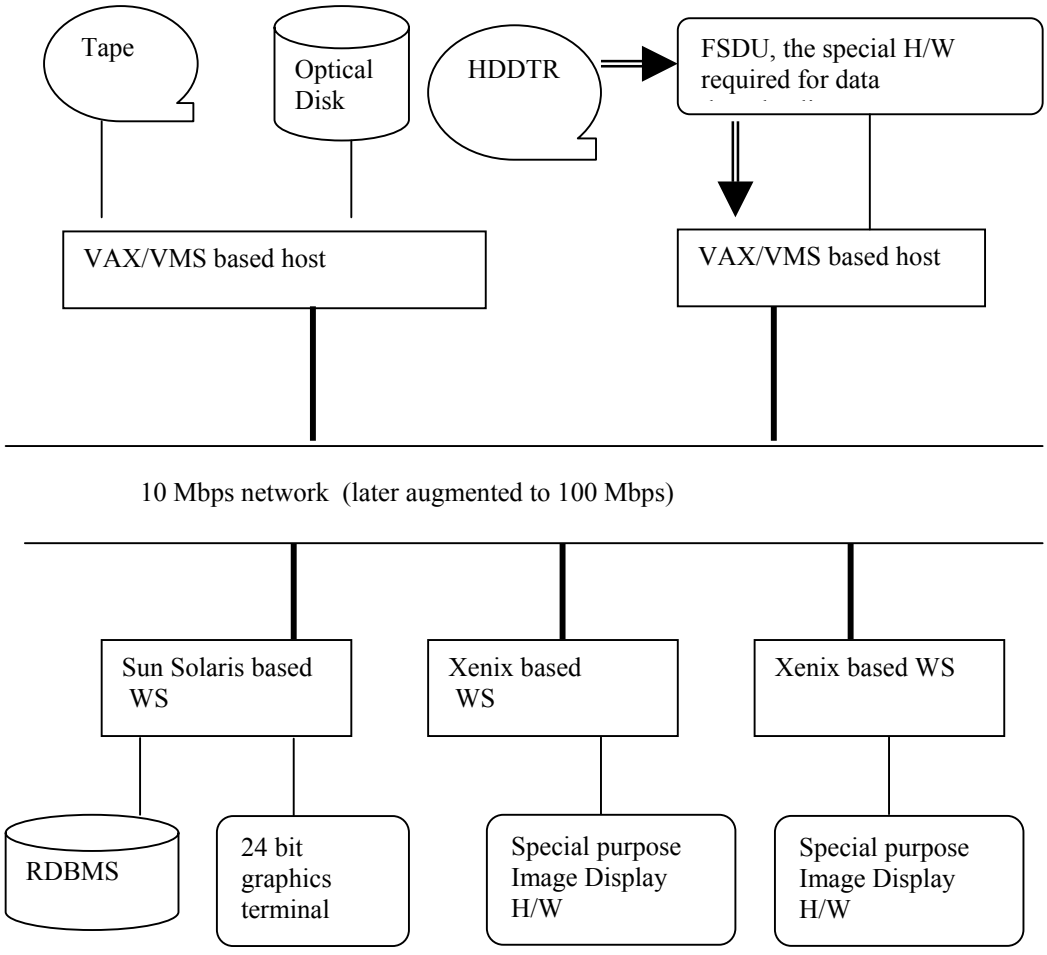



**Figure-3: IRS system configuration**

**Table 2: Details of the packages operationalized for IRS-1C under DEDIP**

| | VAX/VMS | | | | SUN | | | | SIPS | | | | Data Transfer | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Package Name** | **NP** | **Res** | **DS** | **Time** | **NP** | **Res** | **DS** | **Time** | **NP** | **Res** | **DS** | **time** | **Size** | **NF** | **Time** | **Src** | **Dest** |
| LocAcc | **8** | **T,D,P** | **140** | **45** | **2** | **S, D** | **40** | **30** | | | | | **40M** | **5** | **4** | **Vax** | **Sun** |
| | | | | | | | | | | | | | **1K** | **1** | **1** | **Sun** | **Vax** |
| BBR | **3** | **T,P** | **150** | **25** | **3** | **S, D** | **150** | **35** | | | | | **150M** | **6** | **12** | **Vax** | **Sun** |
| | | | | | | | | | | | | | **1K** | **1** | **1** | **Sun** | **Vax** |
| OverLap | **3** | **T,P** | **10** | **9** | **3** | **S, D** | **10** | **12** | | | | | **10M** | **2** | **2** | **Vax** | **Sun** |
| | | | | | | | | | | | | | **1K** | **1** | **1** | **Sun** | **Vax** |
| SideLap | **4** | **T,P** | **10** | **10** | **3** | **S, D** | **10** | **12** | | | | | **10M** | **2** | **2** | **Vax** | **Sun** |
| | | | | | | | | | | | | | **1K** | **1** | **1** | **Sun** | **Vax** |
| RadDqe1 | **4** | **T** | **140** | **45** | **1** | **S, D** | | **1** | **2** | **S** | **150** | **65** | **140M** | **9** | **60** | **Vax** | **Sips** |
| | | | | | | | | | | | | | **1K** | **1** | **1** | **Sips** | **Sun** |
| RadDqe2 | **4** | **T,D,P** | **140** | **45** | | | | | **2** | **S** | **150** | **65** | **140M** | **9** | **60** | **Vax** | **Sips** |
| | | | | | | | | | | | | | **1K** | **1** | **1** | **Sun** | **Vax** |
| GcpUpd | **3** | **D** | **4** | **25** | **1** | **S** | **4** | **30** | | | | | **4M** | **99** | **2** | **Vax** | **Sun** |
| | | | | | | | | | | | | | **2K** | **1** | **1** | **Sun** | **Vax** |
| Vim | **7** | **OD** | **100** | **130** | | | | | **1** | **S** | **100** | **80** | **70M** | **80** | **15** | **Vax** | **Sips** |
| | | | | | | | | | | | | | **1K** | **1** | **1** | **Sips** | **Vax** |
| Wifs | **5** | **OD,F** | **32** | **110** | | | | | | | | | | | | | |
| Swath | **3** | **--** | **30** | **45** | | | | | | | | | **30M** | **100** | **3** | **Vax** | **Vax** |

Where **T**: **T**ape **D**: **D**atabase **S**: di**S**play **P**: **P**rinter **OD**: **O**ptical **D**isk **F**: FSDU
**NP**: **N**umber of **P**rocesses **Res**: **Res**ources required **DS**: Data Size (in MB)
Time: Elapsed time
The data size is given in Megabytes and Kilobytes, time is given in minutes and duration is given in month
The above figures are for one image only.

The resources, time and data transfer requirements shown corresponds to one image (data set). Many images are processed everyday in pipeline mode. The DEDIP supports the pipeline parallel processing configuration, which increases the throughput. The typical example of the pipeline mode, for the Location Accuracy package, is shown in figure 4 for 3 requests. The list of the processes along with the resource requirement for Location accuracy is given in table 3.

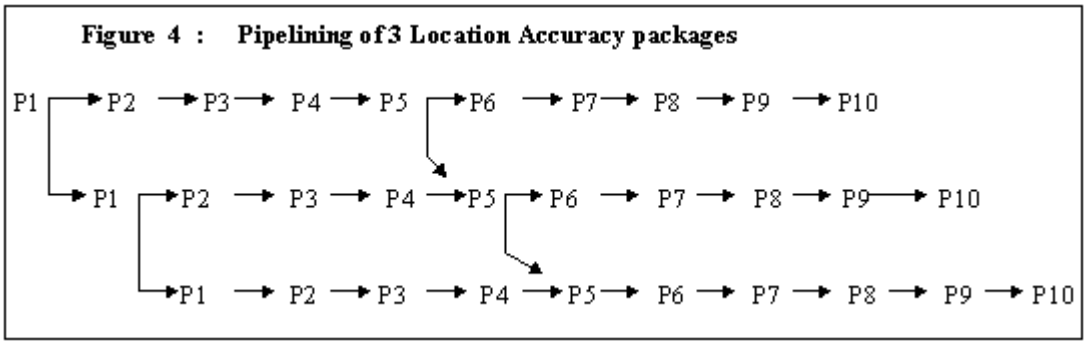



**Table 3 : Package details for Location Accuracy**

| | | | | | *Data Transfer* | | | |
|---|---|---|---|---|---|---|---|---|
| **Sr** | **Process Name** | **System** | **Resources** | **Time** | **NF** | **Size** | **Source** | **Dest** |
| P1 | Ucct DownLoad | Vax | T, 40MB | 25 | 6 | 40MB | Tape | Disk |
| P2 | GCP Extraction | Vax | D, 100MB | 5 | | | | |
| P3 | GCP Approximation | Vax | - | 5 | | | | |
| P4 | File Transfer (DTHS) | Vax/Sun | | 3 | 5 | 40MB | Vax | Sun |
| P5 | GCP Identification | Sun | S, 40MB, Disp | 30 | | | | |
| P6 | File Transfer (DTSH) | Vax/Sun | | 1 | 1 | 1KB | Sun | Vax |
| P7 | Img LtLn | Vax | | 2 | | | | |
| P8 | Parameters | Vax | | 5 | | | | |
| P9 | File Transfer | Vax/Sun | | 1 | 1 | 1 KB | Vax | Sun |
| P10 | DBS Store | Sun | D, S ,100MB | 1 | | | | |

Where NF is number of files to be transferred.

As seen from the table 3 and figure 4, the synchronization of the pipelining is carried out at processes P1 and P5. The process P1 requires tape unit in dedicated mode while P5 requires display screen in dedicated mode to interact with operator. Such pipeline processing is achieved for other applications as well as across applications too.

### 5.3 Utilization analysis (User Feedback)

The simulated test cases as well as the IRS-1C operationalization have proven the DEDIP usage in operations. The DEDIP's main contribution was to relieve the application designers from parallel and distributed programming. The IRS-1C applications designers have used the DEDIP in developing distributed image processing applications. The programmers can also implement process scheduling, file transfer, parallel processing and pipeline processing within their applications. This requires additional time, and specialized skill for each application.

A survey was conducted to quantify the usefulness of DEDIP. Fifteen key-designers were interviewed. Everyone expressed the view that DEDIP is highly required. Furthermore, they were asked to estimate the additional efforts required to develop and operationalize their distributed application in the absence of DEDIP.

- Two computer engineers demanded additional six man months per application.
- Three scientist, having 6 years of experience in developing image processing applications, demanded specialized training and additional 11 man months per application.
- Two project managers demanded dedicated software professionals for the same. ("The additional time and training requirement will be given by the software professionals", they added).
- Eight persons were reluctant to put such complex/tedious efforts. They demanded the DEDIP or an equivalent tool as a mandatory requirement.

If we consider 10 man months per application on an average, DEDIP saved approximately 100 man months efforts for IRS-1C (10 applications). In addition, the need for training & experience for distributed programming became unnecessary. Furthermore, it will save similar efforts for other satellites and image processing applications in future.

## 6. Related Work

In this section, we summarize some of the research efforts that are closely related to our work.

MPI [1] and PVM [2] are the earliest systems to provide the distributed processing environment. SNIPE [3] and Legion [4] are metacomputing systems that can accommodate a heterogeneous mix of geographically distributed high performance machines and workstations. HARNESS [5] extended PVM concept. It removed the PVM kernel limitations to add new services dynamically. HARNESS is built around services of highly customizable & reconfigurable Distributed Virtual Machine.

XPVM [6] is a graphical control interface and a performance monitor for PVM. XPVM supports online & post-mortem monitoring and provides several animated views to monitor the execution of PVM. XPVM does not support the execution of interactive applications. PADE [7] compliments the functionalities that are lacking in XPVM. PADE is mainly devoted to the management of the remote compilation and offers capabilities concerning the PVM configuration. WAMM [8] is also a graphical console that manages PVM and MPI.

JPVM [9], and Java MPI [10] are the Java extension of PVM and MPI respectively. JavaParty [11], ParaWeb [12], Charlotte [13], Popcorn [14], and Javelin [15] are Java based systems for distributed computing using Java. JavaParty provides mechanisms for transparently distributing remote objects. ParaWeb is an implementation of the JVM that allows Java threads to be transparently executed remotely. Charlotte provides high-level solution that decouples programming environment from the execution. Its disadvantage is that the programmer does not have explicit control over resource utilization. However, its eager scheduling enables the runtime systems to efficiently provide load balancing. Popcorn provides a Java API for writing parallel programs for Internet distribution. Applications are decomposed by the programmer into small, self-contained subcomputations, called computelets. The Popcorn is based on buyer-seller concept. It has a centralized entity called market that determines which CPU seller executes the computelet. Javelin is an infrastructure for Internet

based parallel computing. Any free computer system can volunteer to execute a task using the applets supported by the Javelin. It follows a client-broker-server architecture. Bayanihan [16] and Ninflet [17] are also very similar to the Javelin.

All of the above work is excellent. Most of them are complementary to one another, one solving the limitations of the other. Most of them concentrate on providing computation power to a large and complex application efficiently. All of the above work expects efficient parallel and distributed programming skills. Some work [6,7,8,15] addresses the ease of use. However, their definition of ease of use is around application compilation, scalability, progress management, fault tolerance, etc. Furthermore the GUIs of the above models, like XPVM, PADE, support monitoring and controlling a large application in stand-alone mode only. Therefore, they do not require elegant & easy GUI support for simultaneous execution, monitoring and controlling of multiple applications.

DEDIP concentrated on the vast community of users, who are not efficient programmers of parallel and distributed systems. It made the distributed application development very easy. It saves manpower efforts and compromises with little efficiency loss. DEDIP GUI supports all the needs of the operational environment and it executes multiple heterogeneous applications simultaneously.

## 7. Conclusion and Future work

DEDIP proved its heterogeneity, adaptability, smooth operation, operational setup, and ease of operation using IRS-1C operationalization at NRSA. The DEDIP uses the WSManagers on remote systems which do not require any logon to the system and hence can avoid the access rights issues. The efficiency of the DEDIP is visualized during the operation. It is quantified using the simulated test cases. The DEDIP provided a much higher efficiency (223% for parallel application to 106% for sequential application) in comparison with the tasks carried out manually. Furthermore, DEDIP gives a high efficiency of 90-95% compared with the best theoretically expected value (calculated from Figures 2a and 2b).

The IRS-1C image processing application development has proved DEDIP's ease of use. The application designer could save their valuable time and effort while developing their distributed applications under DEDIP. Neither do they need to involve specialist software engineers nor do have they to undergo any training.

Thus the DEDIP optimizes the operational resource requirement, improves throughput and reduces the development & operational cost. Furthermore the DEDIP relieves the image processing scientists from distributed programming burden saving their valuable time and efforts.

DEDIP is being developed in Java to make it truly system independent. Furthermore a few GUIs are also being added to further simplify the application development. The DEDIP is currently following the static scheduling on the application configuration. The DEDIP resource optimization is based on the application configuration. We are working on incorporating dynamic load balancing and automatic resource optimization to achieve better efficiency.

**Acknowledgements**

Mr. SN Dadhaniya is thanked for his implementation help. The IPA designers and operation staff are thanked for their suggestions and help in operationalization.

**References**

1. MPIF, MPI: A message Passing Interface Standard, Final Report, Version 1.1, Message Passing Forum. Available at http://www.mcs.anl.gov/Projects/mpi/standard.htm, June 1995.

2. A. Geist, A. Beguelin, J. Donagarra, W. Liang, B. Manchek, V. Sunderam, PVM- a user guide and tutorials for network parallel computing. Available at http://www.epm.oml.gov/pvm.
3. G. E. Fagg, Keith Moore, Jack J. Dongarra, Al Geist, Scalabale Networked information Processing Environment (SNIPE), Proc. of SuperComputing 97, San Jose, CA, November 1997.
4. A.S. Grimshaw, W.A. Wulf and the Legion Team, the Legion vision of the world wide virtual computer. Communications of ACM, 40(1) (1997).

5. M. Beck, J. Dongarra, G. Fagg, G. Al Geist, P. Gray, J. Kihl, M. Mingliardi, K. Moore, T. Moore, P. Papadopoulous, S. Scott, V. Sunderam, HARNESS: a next generation distributed virtual machine, Future Generation Systems, Vol. 15, (1999), 571-582.

6. J. Kohl, G. Geist, XPVM 1.0 Users' Guide, Technical Report ORNL/TM-12981, Computer Science and Mathematical Division, Oak Ridge National Laboratory, Oak Ridge, TN, April 1995.

7. J. Devaney, R. Lipman, M. Lo, W. Mitchell, M. Edwards, C. Clark, The Parallel Applications Development Environment (PADE), User's Manual, PADE-Major Release 1.4,November 21, 1995.

8. R. Baraglia, R. Ferrini, D. Laforenza, WAMM in the framework of graphical user interfaces for metacomputing management, Future Generation Computer Systems, Vol. 15, (1999), 687-698.

9. A.J. Ferrari, JPVM - The Java Virtual Machine, http://www.cs.virginia.edu/ajf2j/jpvm.html.

10. S. Taylor, Prototype of Java-MPI package, Available at http://cisr.anu.edu.au/sam/java/jav.mpi.prototype.html.

11. M. Philippsen, M. Zenger, JavaParty – trasnperent remote objects in Java, Proc. of ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation, 1997.

12. T. Brecht, H. Sandhu, M. Shan, J. Talbot, ParaWeb: towards world-wide supercomputing, Proc. of 7th ACM SIGOPS European Workshop, 1996.

13. A. Baratloo, M. Karaul, Z. Kedem, P. Wijckoff, Charlotte: Metacomputing on the Web, Future Generation Computer Systems, Vol. 15, (1999), 559-570.

14. N. Camiel, S. London, N. Nisan, O. Regev, The POPCORN project: Dsitributed Computation over the Internet in Java, 6th International World Wide Web Conference, April, 1997.

15. M. Neary, B. Christiansen, P. Cappello, K. Schauser, Javelin: Parallel computing on the internet, Future Generation Computer Systems, Vol. 15, (1999), 659-674.

16. L. Sarmenta, Bayanihan: Web-Based Volunteer Computing Using Java, 2nd International Conference on World Wide Computing and its Applications, March 1998.

17. H.Takagi, S. Matsuoka, H. Nakada, S. Sekiguchi, M. Satoh, U. Nagashima, Ninflet: a Migratable Parallel Objects Framework using Java, Proc. of the ACM 1998 Workshop on Java for High_performance Network Computing, Palo Alto, CA, Feb. 1998.

18. C.V.S. Prakash, A. Shastry, Haresh S. Bhatt, "Schedulers for IRS data products generation", Proc. of Seminar on Supercomputing for Scientific Visualization, BARC, Bombay Feb, 15-17,1994, pp 301-308.

19. J Bruno, EG Coffman & R Sethi, "Scheduling independent tasks to reduce mean finishing time", Comm. of the ACM, Vol. 17, no 7, July 1974, pp 382-387.

20. CVS Prakash et. al., "Scheduler for SIGMA: The image analysis system", Proc. of Int. Conf. on computer communication, Ahmedabad, India, Jan-1993, pp nn71-79.