Weather Forecasting on a Multicluster

M.E. van Casteren

Supervised by

Lex Wolters
Gerard Cats

LIACS

August 31, 2008

Contents

1 Introduction

1.1 Background

The Royal Netherlands Meteorological Institute (KNMI)[1] is the primary organization responsible for weather forecasting in the Netherlands. The forecasting system that they use is called Hirlam: High Resolution Limited Area Model[2]. As the name implies, this system can make detailed predictions, and it is meant for a small region. The model is not designed for the global wrap-around from east to west, and it cannot handle poles because the width between meridians is 0 at the poles. Hirlam operates on a rectangular region of grid points, usually encompassing Europe and Greenland. In addition to initial measurements covering the local region to forecast, the model requires data input for the edges of the region, which comes from a global, lower resolution simulation that is run by a different institute.

1.2 Why this project?

KNMI is interested in the viability of running their forecasts on a cluster or grid, instead of on their current supercomputer. The Leiden Institute of Advanced Computer Science (LIACS)[5] is interested in the performance of their new cluster, the third version of the Distributed ASCI Supercomputer (DAS3)[3]. Specifically, LIACS is interested in the performance of the special LAN and WAN network hardware. These interests are combined in this master project.

1.3 Goal

The goal of this project is to evaluate the performance of the Hirlam weather forecast software on the DAS3 grid. Of special interest is the performance of Hirlam in a multiple cluster situation, so the forecast can be run on a higher number of processors than on a single cluster.

1.4 What has been done?

The Hirlam forecast system has been implemented to run on DAS3, on single or multiple clusters. To evaluate the performance of Hirlam on DAS3, several experiments have been run with Hirlam on the DAS3 grid in different configurations, and the statistics of these tests have been collected and analyzed.

1.5 Overview of this report

This report consists of the following sections:

Hirlam Section 2 gives an overview of the process of running a forecast with the Hirlam software.

Hardware & Middleware Section 3 details the DAS3 hardware, and the supporting software that was used to allow Hirlam to run on DAS3.

Implementation Section 4 describes the process of configuring Hirlam and the supporting software to run correctly on DAS3. It also describes workarounds and scripts that were used in this process.

Experiments & Discussion Section 5 describes the experiments that were run with Hirlam on DAS3, and discusses the results of these experiments.

Conclusions Section 6 contains the conclusions drawn from sections 4 and 5.

Appendix A Appendix A describes in detail how to get Hirlam running on DAS3.

Appendix B Appendix B describes the installation and use of TAU libraries to profile Hirlam, and the TAU `paraprof` program to analyze the profiles.

2 Hirlam

The Hirlam[2] project is a cooperation by the meteorological institutes of the following countries: Denmark, Estonia, Finland, France, Iceland, Ireland, The Netherlands, Norway, Spain and Sweden.

The goal of the project is to develop and maintain a short range weather forecasting system for operational use by all participating countries.

A complete Hirlam run is managed by a software suite called Mini-SMS, a subset of the Supervisor Monitor Scheduler (SMS)[26] suite. Mini-SMS starts, monitors, and cleans up after the various tasks and steps for Hirlam. It collects the logs generated by all the tasks and gathers them in HTML files. This is done according to a task file that describes all the steps to Mini-SMS.

The forecast itself can be executed in a single process, or with many processes in a parallel environment, using the Message Passing Interface (MPI)[15], see 3.4. If the forecast is executed in parallel, the total number of processes is divided into two sets: The Hirlam processes, which do the actual forecast, and the Hirlam Grib Server (HGS) processes, which do the input and output (I/O) for the Hirlam processes.

- The Hirlam processes do the actual weather forecast calculations. The forecast is done in time steps, and the length of each time step depends on the physical distance between the grid points on the world map. Each time step, each Hirlam process needs to communicate with its neighbor processes to transmit information about the edges of its rectangle to its neighbors. This is called local communication.

  Also after each time step, there are a number of tasks that require global communication. The horizontal diffusion of atmospheric conditions[1] requires communication in both the longitude and latitude directions. Statistics collection and synchronization requires communication from all processes to the master process. Additionally, on specified time steps during the forecast, the Hirlam processes communicate their data to the HGS processes, and request new input data from the HGS processes for the boundaries of the map.

- The HGS processes read in the data needed for the forecast, and distribute it to the Hirlam processes. On specified time steps[2], they also collect the forecast data generated by the Hirlam processes and write the forecast state to disk.

2.1 Domain Decomposition

The weather data is represented as a 3D grid with data points that contain information about wind speed, direction, pressure, temperature, etc. for various layers in the atmosphere, and the surface. The grid that is used for all experiments, unless otherwise noted, has 582 data points in the X direction (Longitude), 448 data points in the Y direction (Latitude) and 60 levels in the Z direction (Altitude). For this grid, the forecast uses time steps of 360 seconds, or 6 minutes. The grid is divided into smaller rectangles in the X-Y plane[3], and each Hirlam process does the calculations for its own rectangle. Because the data space to work on is to be divided in 2D, the number of processes Hirlam can be run with must also be defined in 2D. A number of processes for the X and Y direction determine how big the rectangle is for each process. In addition, the number of processes to be used for HGS must be defined. In the rest of this report, these will be referred to as n_x, n_y and $n_{i/o}$

[1] This is done by solving a semi-implicit system with Helmholtz equations, for more information see [27] and [28]. For detailed information about the communication for the diffusion, see [29].

[2] Typically every 3 hours of forecast time

[3] Each rectangle contains the full number of Z-layers.

respectively. The total number of processes used for the forecast is thus $n_x \times n_y + n_{i/o}$. It has been observed[24] that for best performance, the values of n_x and n_y should be close together, and n_x must be smaller or equal to n_y. These observations have been used as guidelines for the experiments of this project.

2.2 Performance Metrics

The most direct way to measure and analyze the performance of Hirlam is to measure the time it takes to complete a certain forecast in different situations. Although Hirlam outputs the time it took to complete the forecast in its logs, this time is not measured accurately enough, and does not include the time spent on the last I/O step. Besides, in a parallel environment Total Time spent on the forecast can be split up into two main groups: Computation Time and Communication Time. To measure accurately the time spent on the forecast, as well as the Computation and Communication time, a software package called Tuning and Analysis Utilities (TAU)[23] has been used.

TAU provides a library that can be linked to the program that needs to be profiled. By itself the TAU library will measure separately the time spent on MPI calls and on the profiled program. Though the MPI calls are part of the program, because they are separately measured, TAU can measure Computation time by excluding the time spent in MPI calls from Total time. Communication Time is calculated for this project by subtracting Computation Time from Total Time. With additional code instrumentation it is possible to measure the time spent on different program functions separately, but this feature was of no interest, since the interest of this project lies with Hirlam's Communication and overall Computation Time.

Aside from the time measured by TAU, there is a derived metric that is of interest: Comparing the time of different experiments to the time of a baseline run gives an idea of the relative speedup compared to that baseline run, how well the forecast scales as more processors and/or clusters are added.

In the experiments for this project, the following metrics have been used:

Total time Measured by TAU as *TAUInclusive*, this is the total time spent in the forecast program, including all MPI calls.

Computation time Measured by TAU as *TAUExclusive*, this is the time spent on only the forecast program, excluding all MPI calls.

Communication time Defined as $\mathit{TAUInclusive} - \mathit{TAUExclusive}$, this is the time spent on MPI communication as a whole.

Relative Speedup A derived metric that can be calculated from either Total, Computation or Communication time. If *BaseTime* is the Total, Computation or Communication time of a chosen baseline run, and *Time* is the time of a different run, then the Speedup of the second run compared to the baseline run can be defined as $\frac{\mathit{BaseTime}}{\mathit{Time}}$.

Individual MPI routines TAU measures the time spent in all individual MPI routines. These routines have names such as MPI_Wait, MPI_Send, MPI_Recv. The times for the most significant of these routines have been used in the analysis.

3 Hardware & Middleware

3.1 DAS3 Hardware

3.1.1 Overview

DAS3 (Distributed ASCI Supercomputer)[3] is a grid that consists of five clusters at four different universities:

- Vrije Universiteit (VU)[4]
- Leiden Institute of Advanced Computer Science (LIACS)[5]
- University of Amsterdam (UvA)[6]
- The MultimediaN Consortium (UvA-MN)[7] - Cluster located at UvA.
- Technical University Delft (TUD)[8]

The clusters were built by ClusterVision[9]. Each cluster has a head node which fulfills the role of fileserver, and a number of compute nodes. The nodes are connected to each other (LAN) and to the other clusters (WAN) via a large switch. The optical network hardware used for this is produced by Myricom[11], and uses an American National (ANSI) Standard called Myrinet[12].

The hardware used for the various clusters differs slightly. The following subsections describe the hardware of each cluster in detail, and subsection 3.1.7 provides a quick table overview.

3.1.2 Vrije Universiteit

The DAS3 cluster at the VU is equipped with Ethernet connections of 1 Gbit/s and 10 Gbit/s, as well as a 10 Gbit/s Myrinet interconnect.

Compute Nodes The VU cluster has 85 dual-CPU, dual-core compute nodes. The processor model is an AMD Opteron DP 280, with a clock speed of 2.4 GHz. Each node has a total of 4 processor cores. Each node also has 4 GB of memory and 250 GB hard drive space.

Head Node The head node at the VU also has a dual-CPU, dual-core AMD Opteron DP 280 processor with a clock speed of 2.4 GHz. However it has 8 GB of memory and a RAID6 storage system with 10 TB of space.

3.1.3 Leiden Institute of Advanced Computer Science

The DAS3 cluster at LIACS is equipped with Ethernet connections of 1 Gbit/s and 10 Gbit/s, as well as a 10 Gbit/s Myrinet interconnect.

Compute Nodes The LIACS cluster has 32 dual-CPU nodes. The processor model is an AMD Opteron DP 252, with a clock speed of 2.6 GHz. Each node also has 4 GB of memory and 400 GB hard drive space.

Head Node The head node at LIACS has a dual-CPU, dual-core AMD Opteron DP 280 processor with a clock speed of 2.4 GHz. It has 8 GB of memory and a RAID6 storage system with 10 TB of space.

3.1.4 Universiteit van Amsterdam

The DAS3 cluster at the UvA is equipped with Ethernet connections of 1 Gbit/s and 10 Gbit/s, as well as a 10 Gbit/s Myrinet interconnect.

Compute Nodes The UvA cluster has 41 dual-CPU, dual-core nodes. The processor model is an AMD Opteron DP 275, with a clock speed of 2.2 GHz. Each node also has 4 GB of memory and 250 GB hard drive space.

Head Node The head node at the UvA has a dual-CPU, dual-core AMD Opteron DP 275 processor with a clock speed of 2.2 GHz. It has 8 GB of memory and a RAID6 storage system with 5 TB of space.

3.1.5 MultimediaN Consortium

The MultimediaN DAS3 cluster is equipped with Ethernet connections of 1 Gbit/s and 10 Gbit/s, as well as a 10 Gbit/s Myrinet interconnect.

Compute Nodes The MultimediaN cluster has 46 dual-CPU nodes. The processor model is an AMD Opteron DP 250, with a clock speed of 2.4 GHz. Each node also has 4 GB of memory and 1.5 TB hard drive space.

Head Node The head node at MultimediaN has a dual-CPU, dual-core AMD Opteron DP 275 processor with a clock speed of 2.2 GHz. It has 16 GB of memory and a RAID6 storage system with 3 TB of space.

3.1.6 Technische Universiteit Delft

The DAS3 cluster at TUD is equipped with Ethernet connections of 1 Gbit/s and 10 Gbit/s, but unlike the other clusters it has no 10 Gbit/s Myrinet interconnect.

Compute Nodes The TUD cluster has 68 dual-CPU nodes. The processor model is an AMD Opteron DP 250, with a clock speed of 2.4 GHz. Each node also has 4 GB of memory and 250 GB hard drive space.

Head Node The head node at TUD has a dual-CPU, dual-core AMD Opteron DP 280 processor with a clock speed of 2.4 GHz. It has 4 GB of memory and a RAID6 storage system with 5 TB of space.

3.1.7 Summary

Table 1 gives a quick overview of the hardware used on the various clusters.

3.2 DAS3 Connectivity

3.2.1 Inter-Cluster

The DAS3 clusters are interconnected via SURFnet[21], a country wide network that connects academic and research institutions with one another. The Starplane [22] project provides the DAS3 clusters with dedicated light paths for intercluster communication, over the SURFnet backbone.

Because the Myrinet protocol is not designed for use in segmented switching networks, currently it is impossible to set up more than two MX links between two different clusters

	VU	LIACS	UvA	UvA-MN	TUD
# Nodes	85	32	41	46	68
# CPUs	2	2	2	2	2
# Cores per CPU	2	1	2	1	1
Clock Speed (GHz)	2.4	2.6	2.2	2.4	2.4
Memory (GB)	4	4	4	4	4
File Server disk space (TB)	10	10	5	3	5
Node disk space (GB)	250	400	250	1500	250
Myrinet	Yes	Yes	Yes	Yes	No
Gbit Ethernet	Yes	Yes	Yes	Yes	Yes

Table 1: DAS3 Hardware.

in an MPI program, since this communication has to go over the Starplane links between the clusters, and there are only two logical links between each pair of clusters. Two direct MX connections in MPI block further connections over the Starplane link, therefore it is currently impossible to run a multicluster program with the MX protocol when the number of required connections is higher than two. Consequently, medium and larger scale multicluster programs have to be run with the TCP/IP protocol, so the Starplane links can be time-shared between all processes.

3.2.2 Internal

All DAS3 clusters except TUD are internally connected by a high speed 10 Gigabit/s optical network and a Myrinet switch. Latency in myrinet mode between the nodes is 2.3 μs.

All nodes also have regular 1 Gbit/s Ethernet lines, connected to stackable Nortel[20] 5530 switches, as a second internal network. The head node has a 10 Gbit/s Myrinet connection to the Myrinet switch and a 10 Gbit/s Ethernet connection to the Nortel 5530 switches. The Nortel switches also service a 10 Gbit/s Ethernet connection to the local university's network, while the Myrinet switch has 8 10 Gbit/s Ethernet connections to SURFnet[21], specifically a Nortel OME 6500. These connections allow fast communication to the other DAS3 clusters.

Figure 1 gives a schematic overview of the internal connections of a DAS3 cluster.

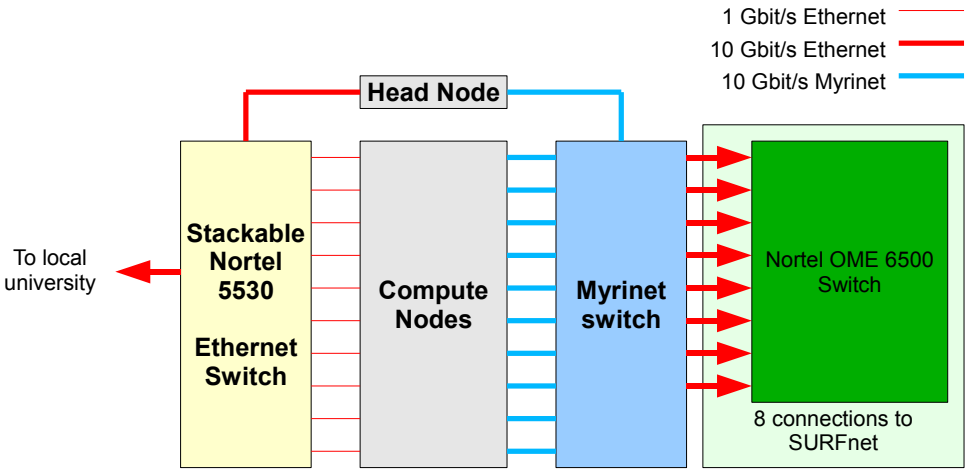

Figure 1: Overview of the internal network connections of a DAS3 cluster.

For standard communication over the Myrinet hardware, the Myrinet drivers encapsulate the default TCP/IP protocol inside the Myrinet eXpress (MX)[12] protocol used on the

optical network. However, programs specially written for the myrinet hardware can also use the MX protocol directly, avoiding the overhead of TCP/IP encapsulation.

3.3 Working with DAS3

All head and compute nodes run Scientific Linux[10] as their operating system. For user home directories and scratch space, the compute nodes have access to the file structure of the head node via the Network File System (NFS). This is transparent for the user; from the user's perspective, all compute nodes and the head node have a shared file system.

3.3.1 General

Users work on DAS3 with a secure shell (SSH) connection to the head node. Aside from the user home directory /home/$user, users also have access to a scratch space located at /var/scratch/$user that can be used as a temporary working directory for large amounts of data.

3.3.2 Reservation System

To run jobs on the compute nodes of a cluster, users are expected to use DAS3's reservation system, the Sun Grid Engine (SGE)[13], to reserve a number of nodes. When the reservation is granted, the job can begin to run on the nodes granted by the reservation.

For further details regarding DAS3's usage policies and instructions, please refer to the DAS3 Usage Policy[14].

3.4 MPI

MPI (Message Passing Interface) is a library specification designed to make it easier to develop parallel programs where separate processes take on different tasks, but need to communicate with each other. There are many implementations of MPI, the most prominent at the time of writing are MPICH[16], MPICH2[17], and OpenMPI[18]. All three are freely available, and open source.

Myricom has ported a special version of MPICH2, MPICH2-MX[19]. This version can take advantage of a low-level implementation of the Myrinet-MX protocol for a cluster equipped with Myrinet hardware. This enables MPICH2-MX to directly use Myricom's own Myrinet eXpress (MX)[12] protocol for MPI communication, thereby bypassing the overhead created by the TCP/IP encapsulation.

For the experiments in this project, MPICH2-MX version 1.0.6 rc1 was chosen to be able to perform experiment runs of Hirlam with both the MX protocol and the standard TCP/IP protocol, so the differences could be compared.

3.5 Pathscale

LIACS has obtained a license for the proprietary PathScale[25] Compiler Suite. Pathscale is a compiler for 64-bit Linux systems, and promises up to 40% performance improvement for compiled programs over other compilers. This compiler has been used to compile MPICH2-MX and Hirlam for the highest possible performance. The exact version as reported by Pathscale is: `QLogic PathScale(TM) Compiler Suite: Version 3.0`.

4 Implementation

To implement Hirlam on DAS3, several challenges and obstacles had to be overcome. This section will describe how these challenges and obstacles were overcome to achieve a successful Hirlam installation on DAS3. For specific details on the scripts and commands used, refer to appendices A and B.

4.1 MPI

Hirlam cannot be run on multiple nodes without an MPI library. DAS3 has an installation of MPICH and OpenMPI available, however an alternative to these options is to manually compile an MPI library in a `local` directory in the user home directory. Reasons for doing this are:

- MPICH cannot be linked to a Pathscale compiled application without being compiled by Pathscale itself.
- MPICH2-MX is a version of MPICH2 patched by Myricom to directly use the MX protocol. This version offers the highest performance with the MX protocol.
- The manually compiled MPI library will be compiled with Pathscale, which will offer increased performance of the MPI libraries during a Hirlam run.

Therefore, MPICH2-MX will be manually compiled for use with Hirlam.

4.1.1 Obtaining MPICH2-MX

MPICH2-MX is only available from Myricom per email request and on submission of a valid Myrinet card serial number. Fortunately the MX libraries at DAS3 include a diagnostic tool that prints information about the Myrinet card of a node, including the serial number.

4.1.2 Compiling MPICH2-MX

Due to the custom setup and Pathscale compiler that will be used, MPICH2-MX requires some preparation to compile correctly.

To allow the use of the TotalView debugger, MPICH2-MX needs to be linked with shared Python libraries. These are not available on DAS3, so Python was configured and installed manually in the `local` directory. A problem with one of the configure scripts in MPICH2-MX prevented it from picking up the Python installation, so this configure script had to be patched. Although the use of the TotalView debugger was not necessary for this project, this information is included for the benefit of future projects. For more information about the use of TotalView with Hirlam, see [30].

MPICH2-MX also needs to link against the MX kernel bypass libraries, however the original libraries installed at DAS3 produced a linker error. After some discussion about the problem, DAS3 administration compiled a special version that worked with the manually compiled MPICH2-MX.

To allow for working multi cluster runs, the recompiled MX library as well as the Pathscale installation were copied to the `local` directory. This is because both the recompiled MX library and Pathscale are only available on the cluster at LIACS. The `local` directory was synchronized to the other clusters after the installation process of MPICH2-MX, Hirlam and TAU was complete.

The main Makefile template for MPICH2-MX used an old form of library inclusion that the Pathscale compiler does not recognize, so this Makefile template also required a patch.

To make it convenient to compile MPICH2-MX multiple times[4], a wrapper script was used. In this wrapper script the installation directory and compilation options can be easily altered. The wrapper script is available in listing 7, appendix A.

After compilation is successful, the only further thing MPICH2-MX needs to function is a secret passphrase for the ring daemons that facilitate the connections between MPI processes. This passphrase is located in a file in the user's home directory with owner-only access rights. In multicluster runs, this file needs to be present on all file servers where Hirlam processes will be run.

4.2 Hirlam

To obtain the Hirlam source code, an account to the Hirlam HexNet site is required. The Hirlam source can be checked out from HexNet's subversion repository with a valid HexNet username/password.

4.2.1 Installing Hirlam

Hirlam's main Makefile was edited to use the architecture most appropriate to DAS3, `linuxgfortran`. However the `linuxgfortran` configuration file required some editing for Hirlam to compile correctly.

Hirlam requires libraries from the HDF[5] toolkit from the HDF group[37]. This library was initially missing, but was later installed by DAS3 administration. The location of this library was added to the `linuxgfortran` configuration file, as well as the inclusion of the szip library that HDF was compiled with.

Furthermore, the `linuxgfortran` configuration file had to be edited to set the compiler to Pathscale and include the necessary MPI and MX libraries. The final version of the `linuxgfortran` configuration file is available in listing 12, appendix A.

In Hirlam's make process, the many fortran files are first preprocessed to new fortran files, which are then subsequently compiled. However unlike the gnu compilers when preprocessing a fortran file, Pathscale does not have an option to save the output code to a new file, it will always be directed to stdout. Therefore this stdout had to be redirected to the desired fortran file, which required a number of patches to Hirlam's makefiles.

After all these preparations were complete, Hirlam could be compiled.

4.2.2 Configuring Hirlam

To run Hirlam, the user must set up an experiment directory. The basic files for this experiment can be "checked out" from the Hirlam source directory with the Hirlam launcher script.

The main configuration file for a Hirlam experiment is `Env_system`. In this configuration file, the following environment variables were (re)defined:

- The `COMPCENTRE` variable was set to `"DAS3"`, which is used in Hirlam's job submission script.

[4]This is useful to create multiple MPICH2-MX installations, one for the MX protocol, and one for TCP. In combination with a symbolic link this allows for quick switching between the MX and TCP libraries.

[5]Hierarchical Data Format

- The scratch directory for Hirlam.
- Variables to control the number of processes Hirlam runs with.
- `HGS_IO_RANKS`, which controls the process ids to be used for HGS processes. The HGS processes are run at the highest process ids by default, which puts them on a secondary cluster in a multicluster run. Hirlam would not run with the HGS processes located on a secondary cluster, so this variable keeps them on the primary cluster in all configurations where the number of nodes on the primary cluster is at least 2.
- The `LAUNCH` variable which influences how Hirlam launches a program.

The job submission script `submission.db` is used by Mini-SMS to submit the various steps in the Hirlam process. This script had to be adapted to work for the DAS3 `COMPCENTRE`, and to use the reservation script. The adapted version for DAS3 is available in listing 16, appendix A.

In the latest Hirlam version, some data preparation for the new Harmonie forecasting model is done during the verification phase.. Due to a bug in the makefile, this part of the verify script will cause Hirlam to fail. This can be solved either by patching the makefile, or disabling this part of the verification phase. Since Harmonie was not used in this project, the verification phase was disabled by modifying the `Env_expdesc` script.

Hirlam cannot make a forecast without input data. Because the process to generate these input files necessary for a forecast on an arbitrary date is rather time consuming, pregenerated files were used for one specific date to forecast. The directory structure that these files need to be in is very specific and had to be deduced through trial and error. The files were prepared in the correct directory structure in the experiment directory. A wrapper script was used that copies the prepared directories to scratch space, then starts Hirlam. The script is available in listing 18, appendix A.

To run Hirlam on multiple clusters, the multicluster reservation script was used. This script is a wrapper around the DAS3 reservation system and is used by `submission.db`. It can ask for reservations on multiple clusters in sequence and will cancel them after the job is finished, but cannot guarantee the reservations on multiple clusters will be granted simultaneously. The script can be edited to control the maximum number of nodes to reserve on each cluster. The script is available in listing 15, appendix A.

The `local` directory that contains all custom libraries required for a Hirlam run had to be synchronized to the other cluster. This was done with `rsync`, an ssh-based remote synchronization tool.

The multicluster script was not designed for two simultaneous MPI runs, however if there were enough nodes available, Mini-SMS would launch the Analysis Postprocessing step and the Forecast step simultaneously, which both use the multicluster reservation script. This would cause Hirlam to crash with no apparent error, so the Mini-SMS taskfile `hirlam.tdf` was edited to make the Forecast depend on the completion of the Postprocessing step. This ensured no two multicluster jobs would be launched simultaneously.

4.3 TAU

The TAU source can be obtained by submitting an email address and a comment to the TAU website[23].

TAU was compiled with a wrapper script similar to the one used for MPICH2-MX. It is available in listing 22, appendix B.

In order for TAU to intercept Hirlam's MPI calls, the TAU libraries had to be linked against Hirlam before the MPICH2-MX libraries, so the `linuxgfortran` configuration file was edited to include the TAU libraries before those of MPICH2-MX.

In multicluster runs, the profile data generated by processes on secondary clusters had to be synchronized back to the primary cluster. The multicluster reservation script was edited to take care of this after the job was finished.

Finally, the TAU `lib` and `bin` directories were added to the environment, a directory to contain the profile data was created and Hirlam was recompiled.

5 Experiments & Discussion

5.1 Questions

To evaluate the performance of Hirlam on DAS3, the primary questions that had to be answered were:

1. How well does the performance of Hirlam scale up when adding more processors for the same forecast? To evaluate this, several experiments with different processor amounts are performed.
2. What is the performance impact on Hirlam when switching from a single cluster to multiple clusters? To evaluate this, a single Hirlam experiment with the same amount of processors is performed on different numbers of clusters.
3. What is the impact of adding more processors to the same forecast, but on a different cluster? This is a more difficult question to answer, because the performance of Hirlam is influenced by the chosen domain decomposition, see 2.1.

5.2 Experiments

5.2.1 Hirlam configurations

Table 2 shows the different Hirlam configurations that were used for the experiments. The Total Nodes column shows the number of physical nodes the entire simulation will be run on. Because each experiment was executed with two HGS processes and two processes per node, Total Nodes is always:

$$\textit{Total Nodes} = \frac{\textit{Hirlam Processes} + 2}{2}$$

Configuration	n_x	n_y	Hirlam Processes	Total Nodes
A	4	6	24	13
B	4	8	32	17
C	6	6	36	19
D	6	8	48	25
E	7	8	56	29
F	6	10	60	31
G	8	8	64	33
H	8	9	72	37
I	12	16	192	97

Table 2: The Hirlam configurations used for the experiments

All configurations used a forecast length of 12 hours. For configurations A through H, the chosen Hirlam domain was: $n_{lon} = 582$, $n_{lat} = 448$, $n_{levels} = 60$, $\Delta t = 360$.

Configuration I uses a different domain: $n_{lon} = 1158$, $n_{lat} = 896$, $n_{levels} = 60$, $\Delta t = 180$.

Configuration	1 Cluster		2 Clusters		3 Clusters		4 Clusters	
	p_1	n_1	p_2	n_2	p_3	n_3	p_4	n_4
A	24	12	12	6	8	4	6	3
B	32	16	16	8	-	-	8	4
C	36	18	18	9	12	6	-	-
D	48	24	24	12	16	8	12	6
E	56	28	28	14	-	-	14	7
F	60	30	30	15	20	10	-	-
G	-	-	32	16	-	-	16	8
H	-	-	36	18	24	12	18	9
I	-	-	-	-	-	-	48	24

Table 3: The number of Hirlam processes assigned to each cluster for runs with 1, 2, 3 or 4 clusters. The columns p_i show the number of processes, the columns n_i show the number of nodes.

5.2.2 Cluster configurations

Table 3 shows how the Hirlam processes of a Hirlam configuration are assigned in runs with one, two, three or four different clusters. The total number of Hirlam processes from table 2 is always equally divided over the clusters that participate in the run. One extra node is always used on the primary cluster for the two HGS processes.

For example, a run of configuration E, see table 2, would need 56 Hirlam processes. The number of nodes required would thus be $(56+2)/2 = 29$ nodes. Suppose this configuration E will be run on 4 clusters, see table 3. Each cluster would run 14 Hirlam processes on 7 nodes, with the primary cluster also running the two HGS processes. Therefore 8 nodes would be reserved on the primary cluster, and 7 each on the three secondary clusters.

Configuration I was a special experiment. The purpose of this experiment was to examine the performance impact of using a very large amount of processors for a forecast. This configuration was run on a grid with four times as many grid points as the standard grid. Also, the timestep length was halved from 6 minutes to 3 minutes. This means the total amount of calculations to be done is 8 times that of the other Hirlam runs. However, the number of processors assigned to configuration I is 8 times that of a 24-processor run, $8 * 24 = 192$, so the amount of forecast calculations per processor is the same as that of a 24-processor run. Therefore the time results of this configuration will be divided by 8, and compared with runs from configuration A to examine the effect of using a very high number of processors for the forecast.

To give an insight into the performance impact of using TCP/IP, the single cluster experiments were run with both the Myrinet-MX protocol and the TCP/IP protocol.

When all these experiments were complete, the timing data from TAU was collected and processed. With gnuplot, graphs were plotted from this processed data.

5.3 Results

In table 3, subsection 5.2.2, the different cluster setups for configurations A through I were shown. For each of these cluster setups, an experiment was run. From the timing results, 6 relative speedup graphs were plotted for Total, Computation and Communication time, for single and multi cluster runs. Two histograms showing the timings for the most significant MPI functions were also produced from the detailed data that TAU has recorded. With these graphs and histograms, the questions of subsection 5.1 can now be answered.

5.3.1 Adding more processors

Figures 2, 3 and 4 show the speedup graphs for Total Time, Computation Time and Communication Time respectively for the single cluster experiments. They will be used to answer question 1 of subsection 5.1: The performance of Hirlam when adding more processors for the same forecast.

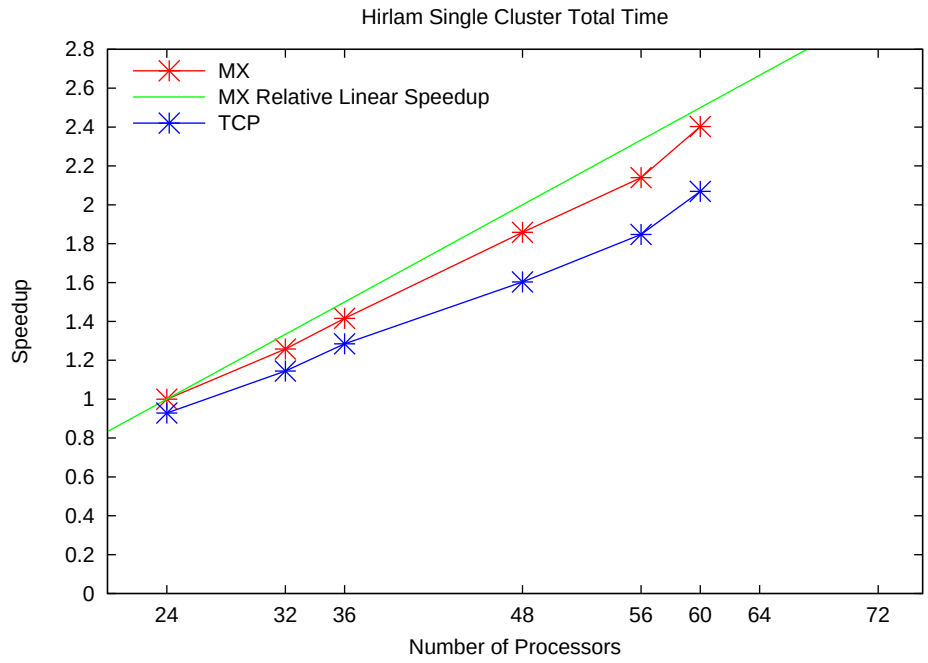

Figure 2: Single cluster relative speedup plot of Hirlam Total Time. The base run used for the y axis and the speedup line is the 24-processor MX run, with a Total Time of 1391 seconds

In figure 2 one can see that the single cluster MX series sinks slightly below the relative linear speedup line with higher processor counts. At 48 processors, MX is at 1.85 speedup, an 85% increase compared to 24 processors, and 92.5% of a linear performance. There is also a gap between the MX and TCP runs. At 24 processors, the TCP run has a speedup of 0.9, or 90% of the performance of the MX run. The 48-processor TCP run reaches a speedup of 1.60, a 74% increase compared to 24 processors, and 85% performance of it's MX counterpart at 1.85 speedup. It appears the gap between the TCP and MX series slightly increases with increasing processor counts.

Figure 3 shows that the Computation Time for both the MX and TCP runs speeds up linearly. The TCP runs are only very slightly below their counterpart MX runs.

In figure 4, one can see that a large gap forms between linear speedup and actual Communication Time with an increasing processor count. The Linear Speedup line in this figure should be interpreted as the ideal Communication Time, if the communication overhead of the forecast were a constant percentage. At 48 processors, MX performs at a 1.48 communication speedup, a 48% increase compared to the 24-processor run. TCP at 48 processors on the other hand has 0.92 speedup, a 23% increase compared to the 24-processor TCP run with 0.75 speedup.

The performance loss of TCP compared to MX is explained by the extra layers of packet and connection handling that are necessary for TCP to function in segmented and potentially unreliable networks such as the internet. MX in contrast is a protocol designed for use in unsegmented and reliable cluster networks, and does not have this extra overhead.

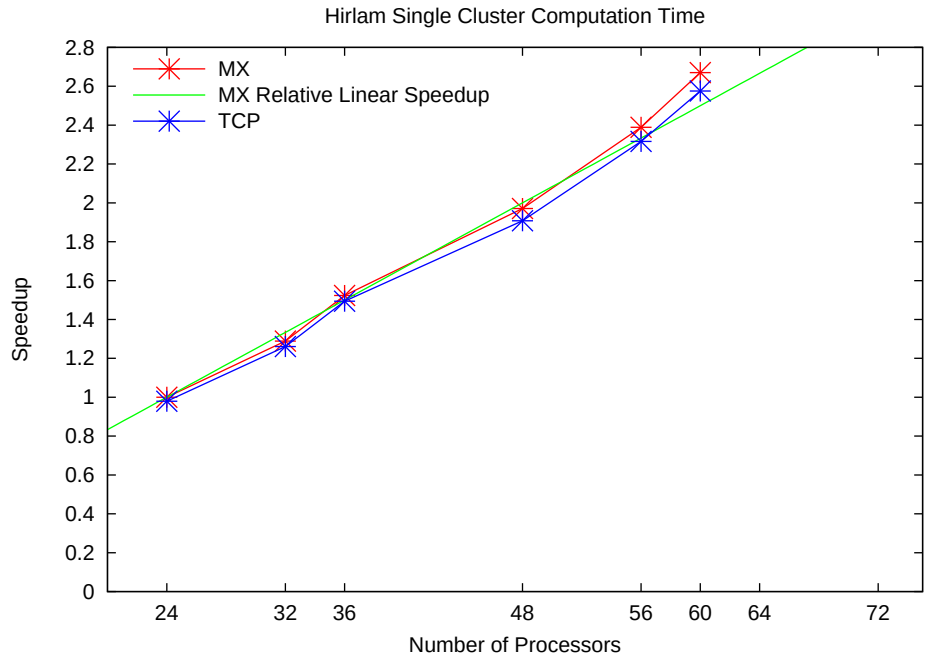

Figure 3: Single cluster relative speedup plot of Hirlam Computation Time. The base run used for the y axis and the speedup line is the 24-processor MX run, with a Computation Time of 1144 seconds.

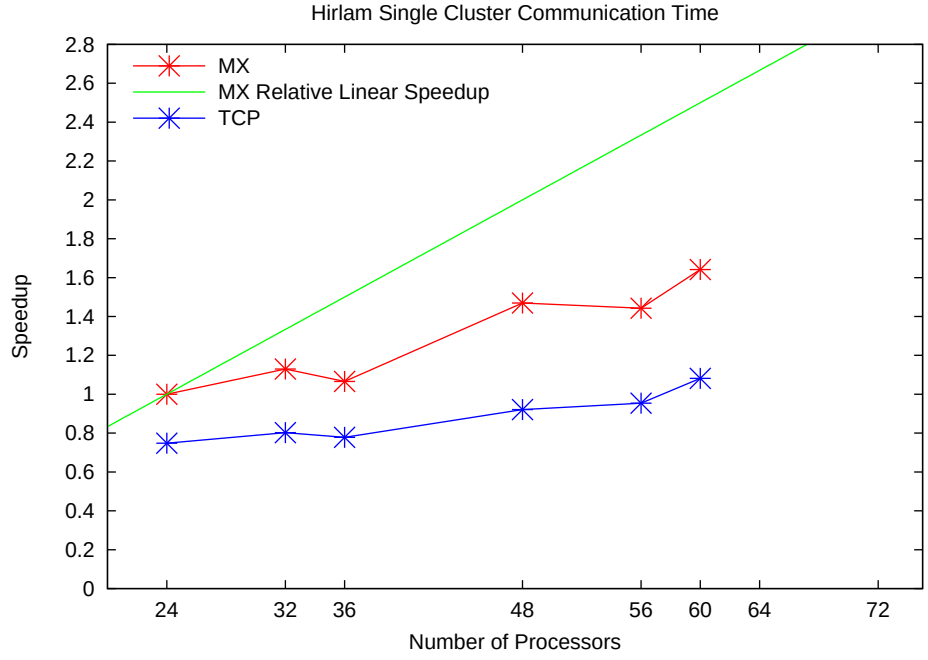

Figure 4: Single cluster relative speedup plot of Hirlam Communication Time. The base run used for the y axis and the speedup line is the 24-processor MX run, with a Communication Time of 247 seconds.

To answer the question of adding more processors, it can now be said that on a single cluster when doubling the number of processors from 24 to 48, Hirlam will gain approximately 85% performance increase with the MX protocol, and 74% increase with the TCP protocol. The efficiency losses by MX and in particular TCP in figure 2 can be attributed to the increased communication overhead. There is no significant loss in efficiency due to scale by the forecast algorithm itself.

5.3.2 Adding more clusters

Figures 5, 6 and 7 show the speedup graphs for Total Time, Computation Time and Communication Time respectively for the multi cluster experiments. They will be used to answer both questions 2 and 3 of subsection 5.1: The performance impact of adding more clusters to a single forecast, and the performance gain of adding more processors to the same forecast, but on a different cluster.

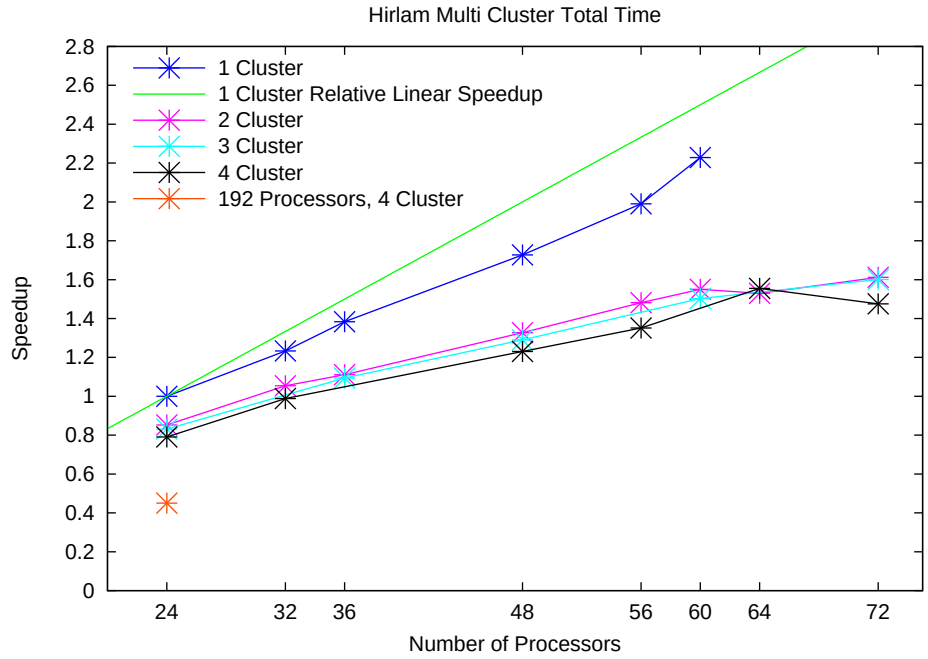

Figure 5: Multi cluster relative speedup plot of Hirlam Total Time. The base run used for the y axis and the speedup line is the 24-processor single cluster TCP run, with a Total Time of 1499 seconds. The time of the 192-processor run was divided by 8, see 5.2.

The multi cluster series in figure 5 have a curve that is not only significantly below the single cluster curve, but also flatter. However, the multi cluster curves are close together. At 24 processors the multi cluster runs have a speedup of 0.79-0.87, 79-87% performance of the single cluster run. At 48 processors the single cluster run gains a speedup of 1.74, a 74% increase compared to 24 processors, and 87% of a linear performance. The 2-cluster run has a speedup of 1.34, an increase of 54% compared to 24 processors, and 77% performance of the single cluster run. The 4-cluster run has a speedup of 1.23, an increase of 56% compared to 24 processors, and 71% performance of the single cluster run.

To formulate an answer to question 3, adding more processors on a different cluster, we will compare single cluster runs with multicluster runs that have a higher number of processors. At 48 processors, the 2-cluster run has a speedup of 1.34, a 34% increase compared to the single cluster 24 processor run. Adding another 24 processors, the 72-processor run

has a speedup of 1.61, a 61% increase compared to single cluster, 24 processors. Looking at 36 processors, single cluster has a speedup of 1.38, and the 72-processor 2-cluster run has a speedup of 1.61, a 17% performance increase.

The 192-processor run is far below that of the comparable 24-processor runs at 0.45 speedup, which is at 57% performance compared to the 4-cluster 24-processor run with 0.79 speedup.

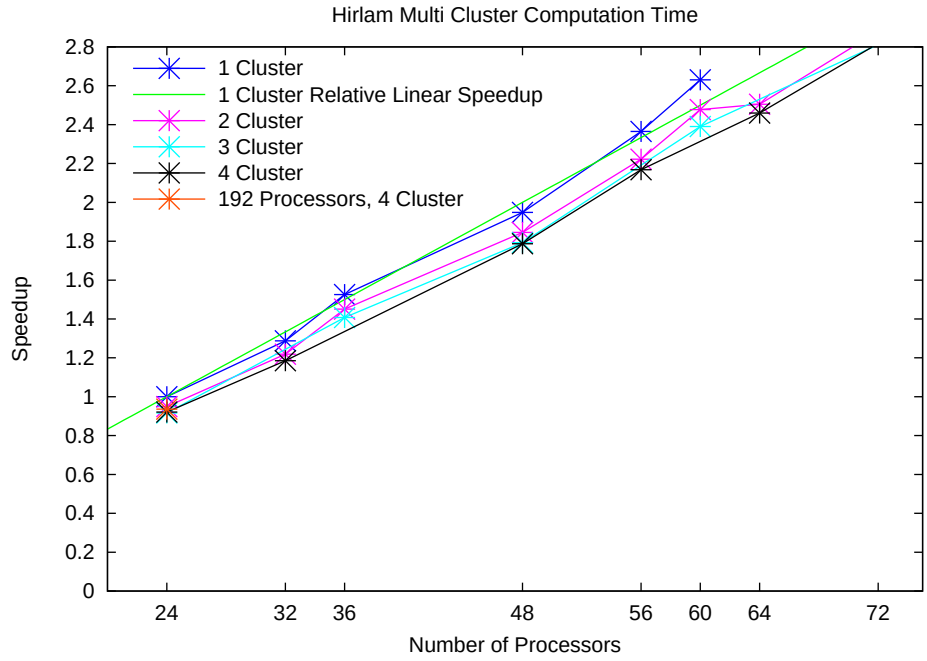

Figure 6: Multi cluster relative speedup plot of Hirlam Computation Time. The base run used for the y axis and the speedup line is the 24-processor single cluster TCP run, with a Computation Time of 1169 seconds. The time of the 192-processor run was divided by 8, see 5.2.

Figure 6 shows that the Computation Time for both the single and multi cluster runs speeds up linearly. The multi cluster runs are slightly below the single cluster runs, but are just as steep as the linear speedup line. The 192-processor run also shows no performance loss compared to the 24-processor runs. This demonstrates that with a larger grid, distribution of the workload over a very high amount of processors is not a problem for the computational part of the algorithm.

The slight loss of performance for the multi cluster runs can be explained by the fact that the single cluster runs were all done at the LIACS cluster, which has the highest processor clock speed. The other clusters have processors approximately 10% slower which becomes visible as a slight performance loss in computation for multi cluster runs.

In figure 7, a large gap forms between linear speedup and single cluster Communication Time with an increasing processor count. There is also a significant gap between the single cluster runs and the multi cluster runs. While Communication Time for 48 processors on a single cluster still gains a speedup of 1.23, the speedup for all multi cluster runs remains between 0.52 and 0.68, showing no significant gain. Also an interesting observation is that even within the Communication Time graph, the multi cluster runs perform very similarly. The special 192-processor run has extremely poor performance, with a speedup of only 0.16.

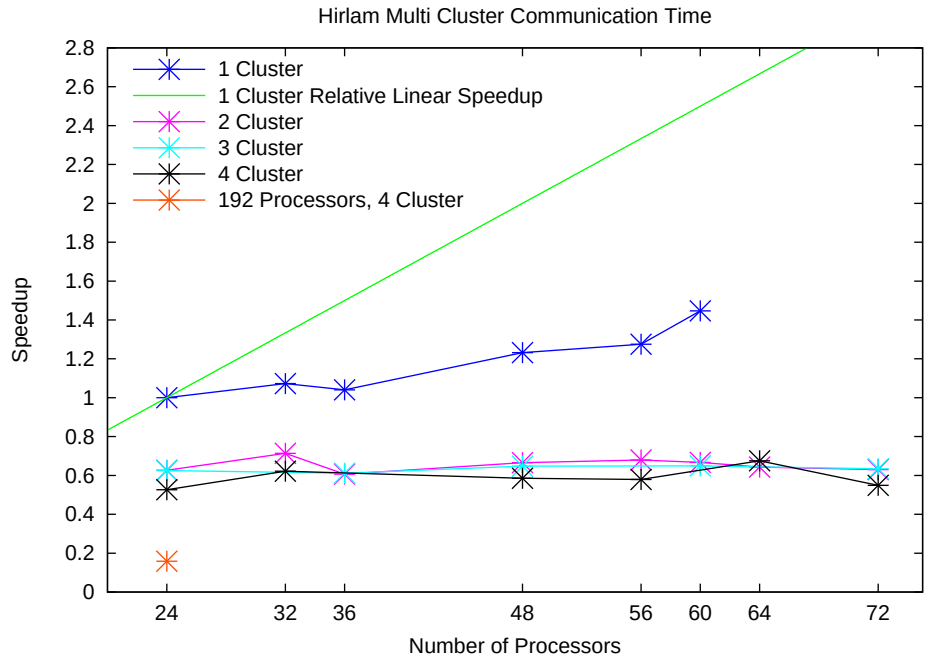

Figure 7: Multi cluster relative speedup plot of Hirlam Communication Time. The base run used for the y axis and the speedup line is the 24-processor single cluster TCP run, with a Communication Time of 330 seconds. The time of the 192-processor run was divided by 8, see 5.2.

To answer the question of adding more clusters for a single forecast, it can now be said that there is a significant performance impact for switching from one cluster to multiple clusters. At 24 processors, Total Time multi cluster performance is 79-87% of single cluster performance, but the performance gap increases with higher processor counts. At 48 processors, Total Time multi cluster performance is 71-77% of single cluster performance. However, there is no significant performance impact for adding more clusters to a forecast that is already multicluster.

To answer the question of adding more processors to a single forecast, but on a different cluster, it can be said that there is less benefit to doing this when the initial number of processors per cluster increases. Adding 24 more processors on a second cluster to a 24-processor run gives a 34% performance increase, and adding another 24 on a third cluster gives a 61% increase. However, adding 36 more processors on a second cluster to a 36-processor run gives only a 17% performance increase, half the increase seen with 24 processors. There is a point where doubling the number of processors onto a second cluster will no longer give a performance benefit. As the 36 processor run shows, this point is already very close.

5.3.3 Communication overhead

From the previous subsections, it is now clear that the computational part of the forecast algorithm scales close to linearly when more processors are added. It is the Communication Time that causes the poor performance for multi cluster runs. Figure 8 and 9 show a stacked histogram breakdown of the time spent by Hirlam processes in the 7 most significant MPI communication functions for the experiments that have been run, in absolute time and in percentages. These figures can give additional insight into which kind of communication increases the most in multi cluster runs.

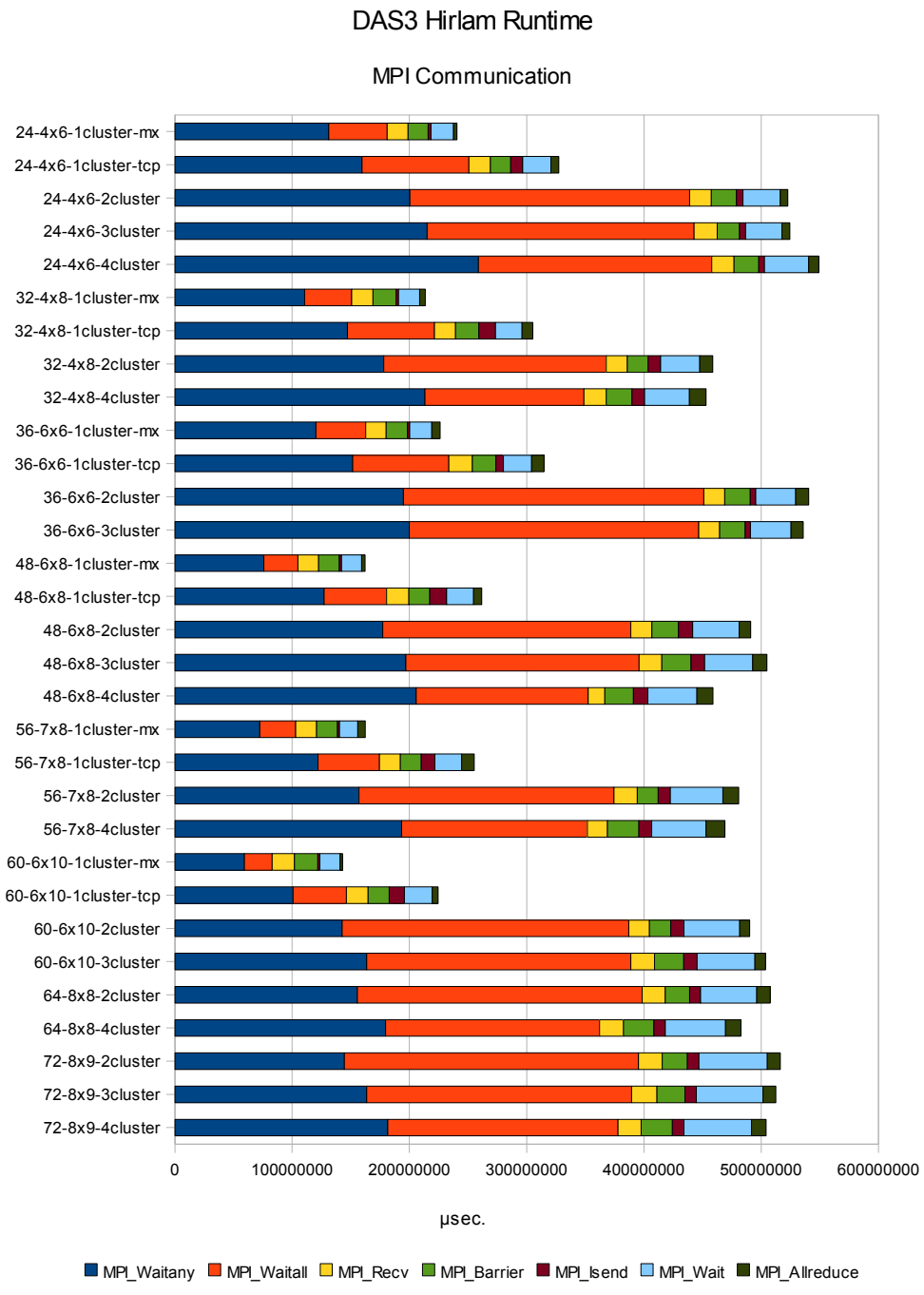

Figure 8: Histogram breakdown of absolute time spent in MPI communication functions. The 192-processor run has been omitted from this figure because it would marginalize the other experiments in the graph too much.

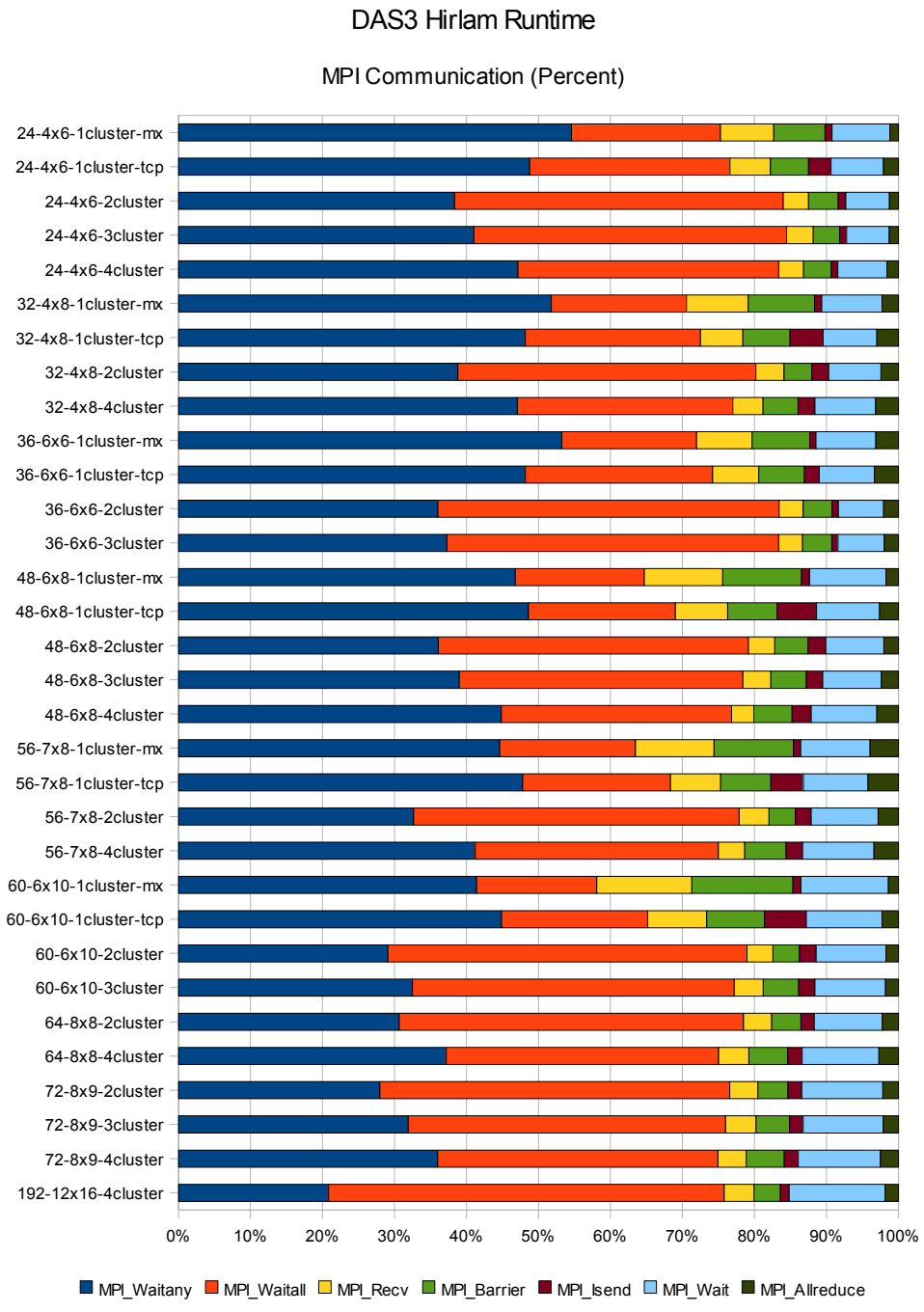

Figure 9: Histogram breakdown of percentage of time spent in MPI communication functions.

Figure 8 shows again that the multi cluster runs spend significantly more time in communication functions than the single cluster runs. However examination of figure 8 reveals a trend that is confirmed by figure 9: The multi cluster runs see an increased amount and percentage of time spent in the MPI_Waitany and MPI_Waitall functions compared to single cluster runs, with the other MPI functions becoming a smaller factor. It can be observed that in multi cluster runs, the MPI functions MPI_Waitany and MPI_Waitall represent the bulk of the increased communication time, while the other MPI functions do not change significantly. When looking at only multi cluster runs, it can also be observed that MPI_Waitany time increases significantly as the number of clusters is increased, and MPI_Waitall decreases significantly.

Examination of the Hirlam source code shows that the local communication step uses only MPI_Waitany calls when waiting for another process. The global communication step uses only MPI_Waitall calls to wait on another process. The HGS communication step uses both MPI_Waitall and MPI_Wait calls to wait for another process, however as figure 8 and 9 showed, the time spent on MPI_Wait does not change significantly for the different runs. These MPI wait calls are not used elsewhere in the code. MPI_Waitall time can thus be attributed to communication of a global nature: HGS and the global communication step. MPI_Waitany time can be attributed to the local communication step.

This information can be combined with the observation that as more clusters are added, MPI_Waitany time increases percentually, while MPI_Waitall time decreases percentually. In other words, when switching from 1 to 2 clusters, both local and global communication increase by a large amount. When increasing the cluster count beyond 2, the local communication step increases again, but the global and HGS communication time decreases.

The Starplane project provides an abstraction from the SURFnet hardware that connects the different DAS3 sites, and on a logical level, every cluster is connected to every other cluster. This means that in two-cluster runs, there are two links in total, in three-cluster runs, there is a total of 6 links, and in four-cluster runs, there is a total of 12.

To help understand the following conclusions, figure 10 depicts how the Hirlam processes in a 24-processor run are spread out across the Hirlam domain, for two, three and four clusters. The differently colored areas represent the different clusters the processes run on. Any time a process of one color communicates with a process of another color, the communication will have to go across the Starplane network.

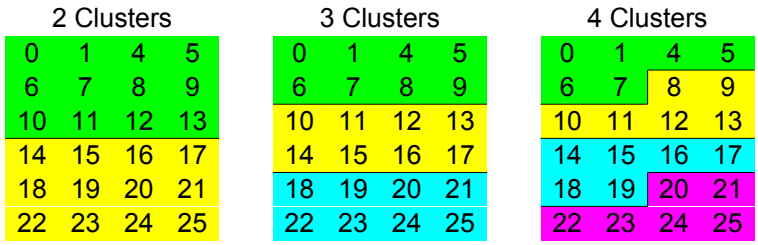

Figure 10: Borders between the clusters for 24-processor runs

Now the following conclusions can be drawn regarding communication overhead:

- Introducing a Starplane link into the network of a Hirlam run (a two cluster run compared to a single cluster run) greatly slows down the Local, Global and HGS communication steps. The two links between the clusters have to be time-shared by all processes for communication. In the figure 10 example, the yellow processes now have to communicate via a Starplane link to the green processes. A plausible explanation for this slowdown due to time-shared links is that TCP communication over the Starplane network goes through a “queue” for encapsulation and transport,

causing increased MPI waiting times. MX connections between two clusters take up an entire link, so these connections do not need to be time-shared, and no queues or encapsulation would be necessary.

- Increasing the number of clusters above 2 increases the number of Starplane links in the network between the nodes. For the Local communication step this is harmful, since there are now more nodes that must transmit information to a neighbor via a Starplane link. In the figure 10 example, the 4-cluster example shows many more borders between the different processes than the 2-cluster example.

  However, in the Global and HGS communication step, there was already a lot of communication that had to cross the Starplane link, because of the All-to-one nature of these steps. Because every cluster is connected to every other cluster, this means that a smaller portion of this All-to-one communication has to go over each individual Starplane link. In the figure 10 example with 4 clusters, yellow processes communicating back to green communicate via a different Starplane link than cyan or magenta processes.

  To summarize, there are more Starplane links to share the burden of communication between all the Hirlam processes, and this relieves part of the load on a single Starplane link during the Global and HGS communication steps. The decrease in global communication approximately cancels out the increase in local communication, and this is why increasing the number of clusters from 2 to 3 or 4 does not significantly change the overall MPI communication overhead. This is likely a coincidence for the small amount of clusters and processors used in experiments A through H, and may change when the number of processors is increased[6].

The Starplane link "queue" explanation given here could not be tested, since administrator access to the cluster and the Starplane hardware would be required to monitor the state of the hardware and the "queues" during an experiment.

[6]For example, the 192-processor run (configuration I) has a different balance between local and global communication than the other runs, as seen in figure 9.

6 Conclusions

In section 2 of this report we explained the Hirlam forecast model and how its performance was measured. Section 3 gave an overview of the DAS3 clusters and supporting software used in the experiments. Section 4 detailed the approach to implementing Hirlam on the DAS3 clusters, for single and multiple clusters. Finally, section 5 discussed the Hirlam experiments, their results, observations about these results and explanations for these observations. In this section we now draw the conclusions from the rest of the report:

- Hirlam performance scales well on a single cluster with the MX protocol. Doubling the number of processors from 24 to 48 gives an 85% performance increase, with 92.5 efficiency of a linear scaling. Using TCP communication gives 92% performance of MX at 24 processors, and 85% performance of MX at 48 processors.
- The computational part of the forecast algorithm is well suited to large scale parallellization. Computation Time performance for runs with higher processor counts scales linearly with the number of processors. The Computation Time result of the 192-processor run compared with the 24-processor runs shows that there is no Computation Time performance loss for very high processor counts if a larger problem is presented.
- The introduction of a second cluster in a Hirlam run causes both the local and global communication step of the forecast algorithm to slow down considerably due to the bottleneck presented by the Starplane link. At 24 processors the multicluster runs have 79-87% of single cluster performance, but at 48 processors the multicluster runs have 71-77% of single cluster performance.
- Although the Communication overhead introduced by switching from a single cluster to two clusters is significant, increasing the number of clusters from 2 to 3 or 4 has no significant performance impact. The increased number of Starplane links in the network slows down the local communication step, but has a mitigating effect on the global communication step. These two factors approximately cancel each other out. In other words, for a fixed number of processes, there is no significant performance difference between a two-, three- or four-cluster experiment.
- Adding more processors onto a second cluster to gain more speed gives less benefit as the initial number of processors increases. Adding another 24 processors on a second cluster to a 24-processor single cluster run gives a 34% performance increase, and adding another 24 processors on a third cluster gives a 61% increase compared to 24 processors. However adding 36 processors on a second cluster to a 36-processor single cluster run gives only a 17% increase. There is a nearby point where doubling the number of processors onto a second cluster no longer gives a speed increase.
- Performance on a single cluster configuration with the MX protocol gains a far greater speedup than that on a multi cluster configuration as the number of processors is increased. Doubling from 24 to 48 processors, MX increases in performance by 85% while multicluster runs increase by 54-56%. At this time multicluster runs have limited scalability in the number of processors, though good scalability in the number of clusters. The limited scalability in the number of processors is due to the bottleneck presented by the Starplane links, compounded by the inability to use the Myrinet-MX protocol in multicluster runs.
- Because the job schedulers on the different clusters are not tied together, a reservation script can never guarantee a reservation slot on multiple clusters at once, starting at exactly the same time. This can lead to problems with starting a multi cluster job when one or more of the clusters is heavily used. However, fixing these problems is not feasible with such a reservation script, and is outside the scope of this project.

6.1 Future work

To improve the scalability in number of processors for multicluster runs, suggestions for future work are:

- Gaining a better understanding of Hirlam communication across the Starplane links. This would involve monitoring the state of the Myrinet hardware during a multicluster run, and finding more information about the way Starplane works, since detailed hardware and software information about the Starplane project is not freely available. This will give a greater insight into the reason for the multi-cluster slowdown compared to single cluster runs, and the reason why 4 cluster performance is not significantly different from 2-cluster performance. The difficulty with monitoring the state of the Myrinet hardware is that this requires root access to DAS3.
- Enabling the MX protocol for multicluster runs. MPICH2-MX failed with MX-runs where more than 2 inter-cluster connections were required between the same pair of clusters. DAS3 System Administration has said they were able to run a multicluster application with the MX protocol with OpenMPI, however there were still severe bugs.

References

[1] KNMI Website - www.knmi.nl

[2] The Hirlam Site - hirlam.org

[3] DAS3 Website - www.cs.vu.nl/das3

[4] Vrije Universiteit, Amsterdam - www.cs.vu.nl

[5] Leiden Institute of Advanced Computer Science, Leiden - www.liacs.nl

[6] Universiteit van Amsterdam, Amsterdam - www.science.uva.nl/research/cs

[7] MultimediaN Consortium, Amsterdam - www.multimedian.nl/en

[8] Technische Universiteit Delft - www.ewi.tudelft.nl

[9] ClusterVision - www.clustervision.com

[10] Scientific Linux - www.scientificlinux.org

[11] Myricom - www.myri.com

[12] Myrinet - www.myri.com/open-specs

[13] Sun Grid Engine - gridengine.sunsource.net

[14] DAS3 Usage Policy - www.cs.vu.nl/das3/usage.shtml

[15] Message Passing Interface - www-unix.mcs.anl.gov/mpi

[16] MPICH - www-unix.mcs.anl.gov/mpi/mpich1

[17] MPICH2 - www.mcs.anl.gov/research/projects/mpich2

[18] OpenMPI - www.open-mpi.org

[19] MPICH-MX - www.myri.com/scs/download-mpichmx.html

[20] Nortel - www.nortel.com

[21] Surfnet - www.surfnet.nl

[22] Starplane Project - www.starplane.org

[23] Tuning and Analysis Utilities - www.cs.uoregon.edu/research/tau/home.php

[24] *Lotte Troen, Korneel Cats, Gerard Cats* - Optimizing the Hirlam forecast model at ECMWF
http://www.hirlam.org/open/publications/NewsLetters/47/Opt.pdf

[25] PathScale Compiler Suite - www.pathscale.com

[26] www.ecmwf.int/products/data/software/sms.html

[27] *McDonald, A.* - The HIRLAM two time level, three dimensional semi-Lagranian, semi-implicit, limited area, grid point model of the primitive equations.
Norrköping, March 1995.

[28] *McDonald, A.* - Default horizontal diffusion coefficient in the reference system.
http://hirlam.org/open/publications/NewsLetters/31/AidanMcDonald.html

[29] *Jan Boerhout* - Reference HIRLAM Scalability Optimization Proposal
http://hirlam.org/open/publications/NewsLetters/44/HIRLAMOptNewsletter.pdf

[30] *Cor Cornelisse* - The Hirlam Weather Forecasting Model on Small Memory Systems
Leiden Institute of Advanced Computer Science, August 2008.

[31] Hirlam Progress Report for EWGLAM 2003 - HGS Introduction
`http://srnwp.met.hu/Annual_Meetings/2003/creports/HIRLAM_EWGLAM_2003.pdf`

[32] Leiden Institute of Advanced Computer Science
`http://www.liacs.nl/`

[33] Dutch ASCI Supercomputer - 3
`http://www.cs.vu.nl/das3/`

[34] TotalView Technologies' main website
`http://www.totalviewtech.com/index.htm`

[35] TotalView Debugger website
`http://www.totalviewtech.com/productsTV.htm`

[36] Description on how to use Hirlam on Linux based machines
`https://hirlam.org/trac/wiki/HirlamHowto/Install/LinuxPc`

[37] HDF-Toolkit website, used to obtain the HDF Toolkit
`http://www.hdfgroup.org`

[38] Keychain, front-end to SSH-Agent
`http://www.gentoo.org/proj/en/keychain/index.xml`

[39] hackinglinuxexposed.com on NetCat-Proxy
`http://www.hackinglinuxexposed.com/articles/20040830.html`

[40] Sun Grid Engine
`http://gridengine.sunsource.net`

[41] Original DAS scheduler, PRUN
`http://www.cs.vu.nl/das3/prun.1.shtml`

[42] Python
`http://www.python.org`

[43] IBM Blue Gene/L - Programming and Operating Environment
`http://www.research.ibm.com/journal/rd/492/moreira.pdf`

Appendix A Hirlam on DAS-3 - Location LIACS

A.1 Introduction

This section will document the process of installing and configuring all the necessary tools to properly run and develop Hirlam on the DAS-3 cluster at LIACS.

Before starting with Hirlam itself, some other topics will be discussed first, since they are mandatory to have Hirlam running successfully on DAS-3. In the process of doing the research required for this thesis a lot of things were stumbled upon, cases in which the local installed applications on DAS-3 simply did not suffice.

Subsection A.2 describes the use of some basic tools to allow easy access to the DAS-3 fileserver node. Subsection A.3 describes the reservation system and the MPI implementation. Finally, subsection A.5 describes the installation and configuration of Hirlam.

A.2 Cluster Access

After an account for DAS-3 is created, it might be handy to setup a few tools making authentication to DAS-3 considerably easier. First the default user password, received by mail, is to be changed, therefore logon to `fs0.das3.cs.vu.nl`, this is sort of the head node for the entire "grid". The password should be changed there, password information on `fs0` will regulary be replicated[7] to the other cluster sites. `fs0` is not reachable from just any IP address, an adress originating from inside the university should be used. Using ssh through an university computer, for example `ssh.liacs.nl`, will also work.

Since most of the time spent on this project involved using a computer outside the university, it was useful to setup ssh-key authentication and a keychain agent. This way it is only necessary to type the password used to decrypt the private key once at system startup, all other authentication is then performed by using ssh-keys. To save some additional trouble on first having to establish an ssh connection to `ssh.liacs.nl` manually, before logging on to `fs1.das3.liacs.nl` a netcat-proxy is used. This simply uses *ssh.liacs.nl* as an ssh relay machine.

A.2.1 SSH identities

First a public/private keypair needs to be generated, this is achieved by running the ssh key generator. A passphrase should be entered and it is strongly suggested to do this. If the private key gets compromised, access to all machines on which the public key has been deployed gets compromised. Therefore it is wise to encrypt the key. After the keypair is generated, the public key should be copied to the machines they will be used on. The process concludes with appending the public key to the authorized_keys file on these machines. For detailed instructions see listing 1.

Listing 1: SSH Public/Private keypairs

```
ssh-keygen -f nameforyourkeypair -t rsa
scp nameofkeypair.pub ssh.liacs.nl:~/.ssh
scp nameofkeypair.pub fs1.das3.liacs.nl:~/.ssh
#On both machines
cat nameofkeypair.pub >> ~/.ssh/authorized_keys
```

[7] This does not happen immediately, if direct access to another cluster site is needed, manually changing the password on that site to is mandatory! However the password set on `fs0` will be the one replicated

A.2.2 Keychain

Keychain[38] is used as a front-end to the ssh-agent handling the key. The advantage of having keychain is, a passphrase only needs to be typed once, at system start-up. Type `keychain nameofkeypair` and add lines from listing 2 to the bashrc script in the root of the user's home directory.

Listing 2: /.bashrc update

```
keychain ~/.ssh/id_rsa
. ~/.keychain/`hostname`-sh
```

A.2.3 NetCat Proxy

The NetCat method, as described in an article at hackinglinuxexposed.com[39] is used to bypass `ssh.liacs.nl`. Step one is to create a shell script containing the proxy, in this case the proxy is nothing more than netcat. To create a netcat-proxy, add the file in listing 3 to the user's homedirectory, and modifiy the ssh config script according to listing 4.

Listing 3: /.ssh/netcat-proxy.sh

```
#!/bin/bash
bouncehost=$1
target=$2
ssh $bouncehost -i ~/.ssh/nameofkeypair nc -w 1 $target 22
```

Listing 4: /.ssh/config

```
Host das3
Hostname fs1.das3.liacs.nl
HostKeyAlias fs1.das3.liacs.nl
ProxyCommand ~/.ssh/netcat-proxy.sh username@ssh.liacs.nl %h
```

To access `fs1`, type `ssh username@das3`, and authentication is handled autonomously in the background.

A.3 Reservation System and MPI

Two scheduler/reservation systems are present on DAS-3. The first and most common one, is the Sun Grid Engine[40]. The second reservation system is a port of the old scheduler used on the original DAS cluster, prun[41].

One would assume the SGE would do the job best, however it is configured in such a basic way that it is rendered useless when it comes to our needs. For example, running a multi-cluster reservation is not possible, local SGE schedulers have no knowledge of the existence of other clusters. The SGE scheduler submits MPICH jobs to the nodes without X forwarding or compatibility with the biggest cluster debugging tool, the TotalView Debugger. All in all, the SGE scheduler is only useful for single cluster production jobs which require no interaction from the user.

Prun does not suffer from these problems, since prun can be used to just reserve a number of nodes for a specified time. Once the reservation is successful, a job ID can be found in its list, and exclusive access to these nodes should[8] be guaranteed. In this case prun is nothing more than a very simple reservation system. Since prun is the only alternative to SGE, it is the only choice. It is quite cumbersome to get an actual reservation,

[8] Although it is against policy, people can run jobs on DAS-3 without using the reservation system.

the `preserve` command used to reserve a number of nodes, directly returns, one has to continuously poll using `preserve -llist` to see if the reservation is actually made!

Next is the MPI implementation, the DAS-3 cluster at LIACS is equipped with a state of the art, super fast, Myrinet 10G network. All compute nodes are fitted with one Myrinet 10G board, tied together with a Myrinet 10G modular switch, leading to super fast, low latency, high bandwidth networking, if configured properly.

DAS-3 has OpenMPI and MPICH installed, however the existing MPICH installation on DAS-3 cannot be linked to Pathscale compiled applications. MPICH requires its libraries to be compiled with Pathscale in order to successfully link to any Pathscale compiled application. This leaves two alternatives:

- Accept the only remaining default, OpenMPI.
- Manually compile an MPICH version.

There is a Myrinet patched version of MPICH2 available from Myricom: MPICH2-MX. This version promises high performance with the Myrinet hardware. A quick test with the Intel MPI benchmark

showed the MPICH2-MX implementation outperforms openMPI[9]. Network performance is of the essence in cluster computing, hence the term High Performance Computing, therefore no second thought to openMPI was given, and MPICH2-MX was used!

The next task is to get MPICH2-MX to compile and install correctly on DAS-3 and creating some decent scripts simulating some kind of reservation/scheduler system on top of preserve.

A.4 Installing MPICH2-MX

Obtaining the Myrinet patched version of MPICH2 is not straightforward, the source is not available directly from their website. If you contact Myrinet, requesting this particular software, they require the serial number from one of their products. On the DAS-3 filesystem the following directory can be found:
`/usr/src/redhat/BUILD/mx-1.2.0e/tools`
This directory contains an executable named `mx_info`. Executing this command results in details on the installed Myrinet board, among this information is the serial number. After supplying this to Myrinet, the source is sent.

Since quite a lot of additional tools are required in order to run HIRLAM, some organization on the installed tools is required. A `local` and `src` directory are created inside the user's home directory. The `src` directory will contain all extracted sources, together with their configuration scripts, installations are done in the `local` directory.

TotalView support is required, one of the prerequisitories of the TotalView debugger is an MPICH2-MX installation compiled with totalview support[10]. To enable TotalView support for MPI, MPICH2-MX needs Python[42] shared libraries. One would expect these to be present on the DAS-3 cluster, but nothing could be further from the truth, the production oriented installation of DAS-3 does not supply shared-libraries for Python, so Python is to be installed manually. Python can be obtained from the Python website, the source is extracted in the src directory. The actual installation is performed by running configure, build and install using the command sequence in listing 5.

[9] Using the Myrinet channel, not using TCP
[10] TotalView provides its own patches to the MPICH community

Listing 5: Configure and make Python

```
./configure --prefix=$HOME/local/python --enable-shared
make
make install
```

It is vital to configure with `--enable-shared`, since the shared libraries are the one thing we require from the installation to compile MPICH2-MX successfully with TotalView debugger support. Once the Python installation is complete, make sure the binary and library path from the custom installation are added to respectively the `PATH` and `LD_RUN_PATH` environment variables in `~/.bashrc`.

MPICH2-MX links against an MX library, installed and compiled by the system administrator at the time of compiling the Myrinet kernel module. The present library, `mx-1.2.0j-2.6.18-clustervision-136.1_cvos` results in a load error at application run time, due to `mx_open_endpoint` symbol issues. After discussing this with the system administrator an alternative library was provided, which was compiled without these additional symbols. This library is located on `fs1` at:
`/usr/local/Cluster-Apps/mx/mx-1.2.0j-2.6.18-clustervision-136.1_cvos-no-compat-syms/lib`

The library is copied to `$HOME/local/mx/lib`. The reason for doing so is because of multi-cluster runs, since only `fs1` has this custom library.

Compilation of MPICH2-MX will still fail due to a bug in one of the configure scripts. This script checks in the wrong directory for the necessary Python libraries, it should check in the manually installed Python installation directory residing in the home directory. This is fixed by editing the file showed in listing 6.

Listing 6: $HOME/src/mpich2-version/src/pm/mpd/configure

```
#Change line number 2738 to contain
if test -f \$HOME/local/python/lib/\$pypgm/config/Makefile;
```

The `Makefile.in` residing in the root of the source directory also requires patching. On lines 103, 106, 110 and 113, the snippet:
`-ldflags ${LDFLAGS} -libs "${LIBS}"`
Should be replaced with:
`-ldflags ${LDFLAGS} ${LIBS}`

To install MPICH2 a wrapper script was written, this script, with minor changes, can be used in consecutive installations as well. To create an MPICH2 installation with the TCP protocol[11], change the installation directory:
`installdir=$HOME/local/mpich2-mx` to `mpich2-tcp` and change the compile option `-with-device=ch_mx` to `-with-device=ch3:sock`. The script can be found in listing 7.

Listing 7: MPICH2 Engage script

```
#!/bin/sh

#
# MPICH2 - Myrinet Configure and install script for DAS-3
# location Liacs (Leiden University) by Cor Cornelisse
#
# based on a script written by Daniel Backlund in Dec2007
# on the Rocks Discussion mailing list from SDSC
#
# This script assumes AMD64 Opteron architecture and
# mx + pathscale software present. Set the correct
# paths below.
#
```

[11] A TCP-compiled version of MPICH2-MX is required for multicluster runs.

```
pathscale=$HOME/local/pathscale
mx=$HOME/local/mx

export PATH=${pathscale}/bin:${mx}/bin:${PATH}

libdir_pathscale=${pathscale}/lib/3.0
libdir_mx=${mx}/lib

export LD_LIBRARY_PATH=${libdir_mx}:${libdir_pathscale}

installdir=$HOME/local/mpich2-mx
mkdir -p $installdir

CC="pathcc"
CXX="pathCC"
CLINK="pathcc"
FC="pathf95"
F90="pathf95"
CFLAGS="-march=opteron -m64 -fno-second-underscore"
CXXFLAGS="-march=opteron -m64 -fno-second-underscore"
FFLAGS="-march=opteron -m64 -fno-second-underscore"
F90FLAGS="-march=opteron -m64 -fno-second-underscore"
LDFLAGS="-Wl,--rpath,${libdir_pathscale} \
-Wl,--rpath,${libdir_mx} -L${libdir_pathscale} \
-L${libdir_mx} -lpscrt"
MX="${mx}"

export CC FC CXX F90 CFLAGS CXXFLAGS FFLAGS F90FLAGS LDFLAGS
export RSHCOMMAND=ssh

./configure --prefix=$installdir \
--with-device=ch_mx \
--with-mx=${mx} \
--with-mpe \
--with-romio \
--enable-cxx \
--enable-compiler-optimizations \
--enable-fast \
--enable-debuginfo \
--enable-g=dbg \
--enable-totalview \
--enable-sharedlibs=gcc > logconfig 2>&1

make > logmake 2>&1

make install > logmakeinstall 2>&1

mv logconfig logmake logmakeinstall $installdir/
cp $0 $installdir/
```

This results in a working installation of MPICH2-MX. Be sure to create the symlink to the `mpich2` directory with `cd ~/local ; ln -s mpich2-mx mpich` for an installation with the MX protocol, or link to `mpich2-tcp` to use TCP. Switching between the two libraries between Hirlam runs can be done simply by removing and recreating the symlink, although for safety Hirlam should be recompiled after a switch. To conclude, the library directory should be added to the `LD_RUN_PATH` environment variable. An example is shown in listing 8.

Listing 8: /.bashrc

```
#LD_RUN_PATH should at least contain the following paths
LD_RUN_PATH=$HOME/local/python/lib:$HOME/local/mx/lib
            :$HOME/local/mpich2/lib
```

The MPICH2-MX daemon, responsible for setting up a communication ring, requires a passphrase for security purposes. A special file is created in the users homedirectory, the rights on the file should be set to read-only for the owner, for example `chmod 600`. For multicluster runs, this file should be copied to all fileservers where the Hirlam processes need to run. The content of the file is shown in listing 9.

Listing 9: /.mpd.conf

```
#Passphrase to secure MPD communication
MPD_SECRETWORD=``yourtopsecretpassphrase''
```

A.5 Compiling & Configuring Hirlam

The first thing to obtain is the Hirlam source code, one needs an account to access the Hirlam HexNet, this account can be used to obtain the source through subversion. With development in mind, it might come in handy to store the source in a separate source directory. Therefore a "Hirlam" directory is created under the "src" directory already present in the user's home directory. The most recent version of hirlam can be found by checking the following URL: `https:///svn.hirlam.org/tags`

The most recent version[12] of the Hirlam forecasting software can be obtained by typing: `svn export https:///svn.hirlam.org/tags/hirlam-7.2rc3`[13]
To allow for more generic scripts and experimenting with different hirlam source trees, a symbolic link is created, named "current" which points to the 7.2rc3 directory. The bashrc file should be updated again, the update can be found in listing 10.

Listing 10: /.bashrc update for Hirlam compilation

```
alias Hirlam=~/src/Hirlam/current/config-sh/Hirlam
export HIRLAMSOURCE=~/src/Hirlam/
```

A.5.1 Compiling Hirlam

Before attempting to compile Hirlam it is wise to check if the system meets the necessary requirements for Hirlam. These requirements can be found on the Hirlam website[36]. The LIACS file-server node of DAS-3 only missed the HDF[14] toolkit from the HDF group[37]. The linuxgfortran configuration file in the Hirlam source directory should be updated to include the correct paths to the HDF toolkit file locations. The actual paths can be found in listing 12, because the HDF toolkit on DAS-3 has been compiled with szip support, the proper szip library needs to be linked in during compilation as well.

Edit the Makefile in the root of the Hirlam directory, and change the "ARCH" variable to "linuxgfortran", this tells Hirlam what architecture to compile for. However linuxgfortran is not entirely consistent with the setup on DAS-3, since Pathscale will be used instead of the GNU C and Fortran compiler.

The Pathscale compiler is unable to just preprocess a file, and then leave it alone. Instead the option to preprocess the .F file and write the output to stdout should be used. There is no way to automatically redirect this to a .f file, the only option left is patching the makefile, redirecting the output to the desired .f file.

The following makefile templates should be adapted:

- makecma.mk
- makehir.mk
- makevar.mk

[12] 7.2rc3 at the time of writing

[13] Using "export" instead of "checkout" will download the source tree without creating all the subversion metadata. Since submitting patches to the repository is not required, "export" is used.

[14] Hierarchical Data Format

Each file contains two lines, which should be adapted to look similar to the line showed in listing 11.

Listing 11: Hirlam makefile patches

```
# Patch 2 instances in the files makecma.mk, makehir.mk
# and makevar.mk

%.o: %.F $(CPP) $(CPPFLAGS_CMA) $< > $(*F).f}

# A SED expression to patch the main/src/Makefile

sed 's/\$< *\([^ ]*\.f\)/\$< > \1/g' <
Makefile > Makefile.new && mv Makefile.new Makefile
```

The Makefile residing in the `main/src` directory also requires patching, however, this involves quite some lines, therefore a SED expression was written, it can be found in listing 11.

The "config.linuxgfortran" file specifies the paths of include directories and libraries needed by Hirlam to compile successfully. The adapted version of config.linuxgfortran to enable compilation with Pathscale and the custom MPICH2-MX installation looks as follows:

Listing 12: /src/Hirlam/current/config/config.linuxgfortran

```
# This file contains the neccessary compiler arguments to compile
# Hirlam on DAS-3 using the pathscale C and Fortran compiler.
#
# This one is for little endian machines, the most common ones:
# Intel Pentium family, Intel Itanium, AMD Opteron, Alpha, VAX.
# For big endian machines, like HP PARISC, IBM POWER or IBM S390,
# SGI MIPS or SUN SPARC,
# change -DLITTLE_ENDIAN into -DBIG_ENDIAN.

# Parallelization
# Possible values -DMPILIB, -DSHMEMLIB, -DMPILIB -DMPI32TO64, <none>
#
PARLIB      := -DMPILIB

# Additional library paths
MPICH2      := $(HOME)/local/mpich2
TAU         := $(HOME)/local/tau
MX          := $(HOME)/local/mx
# Define MPI include and library paths
MPIINCLUDES := -I$(MPICH2)/include
MPILIBDIRS  := -L$(MPICH2)/lib -L$(MX)/lib
MPILIBS     := $(MPILIBDIRS) -lmpich -lmyriexpress -ldl -lrt -lssl
#-fmpich

# TAU variables
TAULIBPATH  := -L$(TAU)/x86_64/lib
TAULIBS     := $(TAULIBPATH) -lTAU -lmpichcxx

# HGS
# Possible values: IPC,MPI,NONE
#
HGS_TYPE := MPI

# HDF definitions correct for Debian GNU/Linux.

HDFPATH     := /usr/local/hdf/hdf4.2r1.amd.64bit
HDFINCLUDES := -I$(HDFPATH)/include
HDFLIBPATH  := -L$(HDFPATH)/lib -L/usr/local/hdf/lib
HDFLIBS     := $(HDFLIBPATH) -lmfhdf -ldf -ljpeg -lz -lm -lsz

LAPACKLIBS  := -llapack -lblas

FFT         := -DFFTW3
FFTPATH     := /usr/local/Cluster-Apps/fftw/gcc/64/3.1.1
FFTINCLUDES := -I$(FFTPATH)/include
```

```
FFTLIBPATH  := -L$(FFTPATH)/lib
FFTLIBS     := $(FFTLIBPATH) -lfftw3

MACHINECPP := -DLINUX -DLITTLE_ENDIAN -DHIRLAM -DPREC32 -DUSEWALLTIME \
              -DHAS_BLAS -DHAS_LAPACK -DGRIB32 $(MPIINCLUDES) \
              $(FFTINCLUDES) $(HDFINCLUDES) -DTIMING
CRAYDEF := -DNONCRAYF

CPP     := pathcc -traditional -P -E
CC      := pathcc
CCFLAGS := -march=opteron -g -O2 $(MACHINECPP)
# $(MPILIBS) $(HDFLIBS) $(TAULIBS)

FC      := pathf95 -march=opteron -fno-second-underscore

#-----------------------------------------------------------------------
# FCFLAGS: single precision reals ->double  but double remain as double;
#          inline compile directives recognised (!OPTIONS);
#          module information file is .mod - sought under -I dir;
#          optimized;
#          Fortran 90 language (-X9);
#          Fixed source form of Fortran 90 (-Fixed).
#          Checks for conformity with F90 standard (-v9).
#          Recursive functions allowed ??
#
# gfortran: -ffixed-form => fixed form;
#           -ffree-form  => free format (F90);
#           -fdefault-real-8 -fdefault-double-8
#               => single precision reals->double; double unchanged
#-----------------------------------------------------------------------

LINK_DEBUG :=

FCFLAGS := -g -O2 #-ffpe-trap=invalid,zero,overflow

FCFLAGS_CMA   = $(FCFLAGS) -module $(ROOTDIR)/$(ARCH)/cmamod -ffixed-form -r8
FCFLAGS_CMA90 = $(FCFLAGS) -module $(ROOTDIR)/$(ARCH)/cmamod -freeform -r8
FCFLAGS_VAR   = $(FCFLAGS) -module $(ROOTDIR)/$(ARCH)/varmod -ffixed-form -r8
FCFLAGS_VAR90 = $(FCFLAGS) -module $(ROOTDIR)/$(ARCH)/varmod -freeform -r8
FCFLAGS_HIR   = $(FCFLAGS) -module $(ROOTDIR)/$(ARCH)/hirmod -ffixed-form
FCFLAGS_HIR90 = $(FCFLAGS) -module $(ROOTDIR)/$(ARCH)/hirmod -freeform

# xlf (IBM/AIX), ifort (Intel/Linux) (and others) always produce
# a lower-case module name; GNU Fortran does too.

MODNAME = $(shell echo $(*F) | tr "[:upper:]" "[:lower:]")
MODEXT := mod

LD          := $(FC)
LD_MPP      := $(FC)

LDFLAGS     := -g

LDFLAGS_VAR := $(LDFLAGS) $(MPILIBS) $(LAPACKLIBS) $(FFTLIBS)
LDFLAGS_CMA := $(LDFLAGS) $(MPILIBS) $(LAPACKLIBS)
LDFLAGS_HIR := $(LDFLAGS) $(MPILIBS) $(LAPACKLIBS) $(FFTLIBS)

AR      := ar
ARFLAGS := rv
MV      := mv
RM      := rm -f
MKDIR   := mkdir
RMDIR   := rmdir
```

A.5.2 Configuring Hirlam

It is time to setup an experiment directory, so create the new directory `$HOME/hl_home/experiment`. To fill this directory with some of the necessary Hirlam basic files the following command is issued:

```
Hirlam setup -d $HIRLAMSOURCEh -r current -h LinuxPC
```

Inside the new Hirlam experiment directory resides a file called `Env_System`, this file contains information on where Hirlam can find its files. An adapted version which should suffice on DAS-3 is shown in listing 13.

Listing 13: Env_System

```
#<title>config-sh.LinuxPC: Describe simple LinuxPC hard/
#software system</title>

# 1.  Identify and describe your system

export COMPCENTRE
COMPCENTRE=DAS3         # computer centre
export SCRATCH
SCRATCH=/var/scratch/$USER

# 1.0 define hosts and the location of this file on each host

# lvfs: large volatile file system, should be fast
HOST0=`hostname` export HOST0
# : seperated list where the system will be needed
HOST_INSTALL=0
# name of a large, volatile, file system on HOST 0
lvfshost0=$SCRATCH
export HOST_INSTALL

SMSHOST=${SMSHOST-$HOST0}       # default is to run on HOST0

# 1.1 Set system hard/software

export LSERIAL NHORPH NAMLIS_E MAXFACT235711

# .false. for non-vector pc, .true. for vector
LSERIAL=.false.
# vectorlength, don't use on non-vector machines
NHORPH=
# string to end a namelist
NAMLIS_E='/'
# maximum no. of factors in FFT factorization
MAXFACT235711="99,99,99,0,0"

# 1.3 Root directories of HIRLAM experiments

# mass storage system. Access by script Access_lpfs
export HL_ARC
# Archive root, in LPFS
HL_ARC=$SCRATCH/hl_arc
# home of experiments runs, in FPFS
home=$HOME

# 2.  Set HIRLAM variables, and, if for installation,
#     extract export system

# 2.1 HIRLAM system location

export HL_REF_FS HL_SDOC HL_DOC HL_HTML
export HIRLAM_CONFIG
export HL_CLDATA HL_CLDATA_MODE

HIRLAM_CONFIG=linuxgfortran     # GNU-make (FFLAGS, etc)
# location of climate files directories
HL_REF_FS=$SCRATCH
# directory with (support) information
HL_SDOC=
HL_DOC=https://hirlam.org/UG/HL_Documentation # HIRLAM documentation
# climate data directory
HL_CLDATA=$SCRATCH/hl_cldata/dat
# mode of the that directory
HL_CLDATA_MODE=a+rx

HL_HTML=$hl_home/hirlam-doc     # directory for html documentation

# 2.3 Experiment identifier and directories

export HL_DATA HL_EXP HL_LIB

EXP=${EXP?"Give 'experiment' identifier EXP \
before running this script"}
HL_DATA=$lvfshost0/hl_home/$EXP # HL_DATA on main HIRLAM host

# dir for scripts,  objects, executables
HL_LIB=$HL_DATA/lib
export BUFRTAB_DIR
BUFRTAB_DIR=$HL_LIB/data/bufrtables     # directory with BUFR tables
```

```
HL_EXP=$HL_ARC/$EXP             # experiment archive directory in LPFS
hl_CMODS=$home/hl_common_mods   # user mods to all exps, in FPFS

# 3.  Path

export PATH

# add HIRLAM to the path

PATH=$PATH:$HL_LIB/$HIRLAM_CONFIG/bin:$HL_LIB/scripts

# 4. Utilities

# 4.1 for the main unix platform

export MAKE MKDIR WHENCE
MAKE=make
MKDIR="mkdir -p"
WHENCE=which

# 4.2 For (mini-)SMS

export SMSMETER CDP
SMSMETER=${SMSMETER-$HL_LIB/scripts/smsmeter}
CDP=${CDP-$HL_LIB/scripts/sms_cdp}

export PERL_TK          # Perl with Tk
PERL_TK=perl

# 5.  Preparative actions

# 6.  Operating system dependants

# 6.1 To run scripts


# 6.2 System libraries

export ECLIB SCILIB
# ${EMOSLIB-~emos/lib/libemos.R32.D64.I32.a}
# probably never used!
ECLIB=
# /usr/lib/libveclib.a
# libs with BLAS/EISPACK, space separated
SCILIB=
# only used if relevant cpp options are set

# 6.3 HDF

export HDFINCLUDE HDFLIBS
if [ $COMPCENTRE = ECMWF ]; then
# done in config.ibmecmwf
#  export _HDF_=_HDF_   # need an underscore when invoking HDF routines
#  ilp=LP64
#  hdf=/usr/local/lib/hdf/HDF4.1r5_${ilp}_extname
   : continue
else
# better be done in the appropriate configuration file
#  hdf=
#  HDFINCLUDE=-I$hdf/include
#  HDFLIBS="-L $hdf/lib -lmfhdf -ldf -ljpeg -lz"
#  export HDFINCLUDE HDFLIBS
   : continue
fi

# 6.4 MPI

export NODES=29         # The number of nodes to reserve.
export TASKS_PER_NODE=2 # Defines the number of processes per node.
export NPROCX=7         # Processes in the X direction
export NPROCY=8         # Processes in the Y direction
export NPROC_HGS=2      # Processes to be used for I/O

# For Cor Cornelisse's implementation of HGS on low memory systems
#export HGS_MASTERS=2   # Number of master HGS processors
#export HGS_SLAVES=2    # Number of slaves per master

# An increasing list of the process ranks you want to use for
# the HGS processes. There must be at least NPROC_HGS items here!
# This is useful to control on which cluster the HGS processes
```

```
# will run in a multi cluster run.
#export HGS_IO_RANKS="1 2"

export LAUNCH="mpiexec -tvsu -np $[$NODES * $TASKS_PER_NODE]"

# 6.5 mini-SMS

export PERLTKDIR                # the directory of the Perl-Tk toolkit
PERLTKDIR=
export JOBOUTDIR                # directory of job stdout/stderr files
JOBOUTDIR=$HL_DATA

# 6.6 Asynchronous I/O (HIRLAM Gribfile Server)

export HGS_TYPE
HGS_TYPE=MPI            # NONE,IPC,MPI

# 8.  Derived environment variables

# 9.  Obsolescing

export HL_COMDAT
HL_COMDAT=$HL_DATA/..

# 10. Debugging
# To enable debugging, uncomment the following line:
# export HGS_DEBUG=2
```

To create an actual reservation on the reservation system a special script was developed. The script is a wrapper around prun and the manually installed MPICH2-MX version. Two versions are availabe, one for single cluster runs, another for the use of multicluster runs, in listings 14 and 15 respectively. A symbolic link called reserve.sh, which points to either of the two versions, is created. The symbolic link, along with the scripts are stored in the `scripts` subdirectory of the experiment. This directory might need to be created if it is not there yet, everything contained in the experiment directory is copied to scratch at run time. This takes care of having the reserve script among other scripts available everywhere, though the reserve script will only be used by one file server, which is the one used to start Hirlam.

Listing 14: reserveSingle.sh

```
#!/bin/sh

#
# Script to bypass SGE and launch job manually after reserving
# nodes properly through the reservation system.
#
# It will use preserve to make a reservation and then wait
# untill these nodes become available for use. Once they are
# available the MPD (MPICH2) daemons will be started on
# all compute nodes after which the actual job is launched. Once
# it finishes, or the reserved time elapses the MPD daemons
# will be shutdown and the reservation is cancelled
#
# By Cor Cornelisse - LIACS (Leiden University)
#

# First parameter is number of nodes
# Second parameter is the number of tasks (CPUs) per node
# Third parameter is the Number of HGS Data Processors
# Fourth parameter is the Environment System of Hirlam
# Fifth parameter is the actual program to be executed
# Sixth parameter is where the stdout should be redirected to

cpusPerNode=$2
totalCPUs=$((${1}*${2}))
dataProcessors=$3
Env_system=${4}
machineFile=/var/scratch/$USER/machineFile
mpiRunCommand=${5}
logFile=${6}
experimentName="experiment"

echo Number of processors to be reserved: $totalCPUs
```

```
# Reserve a given number of nodes for a specified time
reserve=`preserve -e ~/nodelist -1 -# ${1} -t 00:15:00`
# Grep our reservation ID
reservationID=`echo $reserve | sed \
's/Reservation number \([0-9]*\).*/\1/'`
echo Reservation ID = $reservationID

machineFile=$machineFile$reservationID

# Now we'll start waiting for the nodes to get assigned to us
echo -ne Waiting
# Store nodes in nodeList once the reservation is made
nodeList="-"
while [ "$nodeList" = "-" ]; do
        nodeList=`preserve -llist | grep $reservationID | sed \
        's/.*[0-9][0-9]:[0-9][0-9]\t[a-z]*\t[\0-9]*\t\(.*\)/\1/'`
        echo -ne .
done

echo $'\n'$'\n'The Following nodes have been reserved and await \
further instructions: $nodeList

nodeSuffix=".das3.liacs.nl"

# Time to create a machinefile
touch $machineFile
for i in $nodeList;
do
        echo "$i$nodeSuffix" >> $machineFile
done;

echo Machine file created, displaying contents:
cat $machineFile

# Set the correct start node (first node in the nodeList)
set $nodeList
startNode=$1$nodeSuffix
echo Starting from: $startNode

# Remove first node from machineFile (otherwise it will try to start
# MPD twice on the same node and fail).
sed -i '1d' $machineFile

# The Nodes ar reserved for our use, let's launch our own MPD daemons
ssh -Y $startNode 'mpdboot --totalnum='$((${totalCPUs}/2))' \
--file='$machineFile

# Check our MPD ring is established and working
ssh -Y $startNode mpdtrace

# Make a copy of the original Environment since we are going to modify
# the launch parameter, and would like to restore this after the
# run
echo Environment file: $Env_system

# Add the nodelist to the mpirun command
# Be carefull with this sed expression, it looks for "-np $[" in
# the LAUNCH variable present in the Env System. This is defined in
# the hl_home/config/config.LinuxPC. It adds "--debug and the machine
# list
sed 's/\(\-n \$\[.*]\)/\1 \-env NPROC_HGS '$dataProcessors'/g' \
$Env_system > /var/scratch/$USER/hl_home/$experimentName/lib/Env_system

# Let's ROCK
echo Launching: $mpiRunCommand
echo Logfile is $logFile
ssh -Y $startNode 'source ~/.bash_profile && '$mpiRunCommand'' \
> $logFile
echo DONE....

# We are done, shutdown MPD daemons on the used nodes
ssh $startNode mpdallexit

# Restore original Env_system
cp $Env_system \
/var/scratch/$USER/hl_home/$experimentName/lib/Env_system

# Remove the reservation
preserve -c $reservationID
```

```
rm $machineFile
```

Listing 15: reserveMulti.sh

```
#!/bin/bash

#
# Script to bypass SGE and launch job manually after reserving
# nodes properly through the reservation system.
#
# It will use preserve to make a reservation and then wait
# untill these nodes become available for use. Once they are
# available the MPD (MPICH2) daemons will be started on
# all compute nodes after which the actual job is launched. Once
# it finishes, or the reserved time elapses the MPD daemons
# will be shutdown and the reservation is cancelled
#
# By Cor Cornelisse
# and Maarten van Casteren
# LIACS (Leiden University)
#

# First parameter is number of nodes
# Second parameter is the number of tasks (CPUs) per node
# Third parameter is the Number of HGS Data Processors
# Fourth parameter is the Environment System of Hirlam
# Fifth parameter is the actual program to be executed
# Sixth parameter is where the stdout should be redirected to

cpusPerNode=$2
totalCPUs=$((${1}*${2}))
dataProcessors=$3
Env_system=${4}
machineFile=/var/scratch/$USER/machineFile
mpiRunCommand=${5}
logFile=${6}
reservationTime="01:30:00"
mpdListenPort=37832
experimentName="experiment"

# Make a copy of the original Environment since we are going to modify
# the launch parameter, and would like to restore this after the
# run
echo Environment file: $Env_system

# Add the nodelist to the mpirun command
# Be careful with this sed expression, it looks for "-np ${" in
# the LAUNCH variable present in the Env System. This is defined in
# the hl_home/config/config.LinuxPC. It adds "--debug and the
# NPROC_HGS environment variable.
sed 's/\(\-np \$\{.*]\)/\1 -env NPROC_HGS '$dataProcessors'/g' \
$Env_system > /var/scratch/$USER/hl_home/$experimentName/lib/Env_system

numberOfClusters=0
multiClusterRun=1
function declareCluster {
        numberOfClusters=$((${numberOfClusters}+1))
        fileServers[${numberOfClusters}]=$1
        suffices[${numberOfClusters}]=$2
        maxNodes[${numberOfClusters}]=$3
}
# Declare new clusters as below. NOTE: The first cluster defined must
# be the cluster from which the program should be started!
#declareCluster "fileserver.suffix.com" ".suffix.com" maxNumberOfNodes
declareCluster "fs1.das3.liacs.nl" ".das3.liacs.nl" 25
declareCluster "fs0.das3.cs.vu.nl" ".das3.cs.vu.nl" 24
declareCluster "fs2.das3.science.uva.nl" ".das3.science.uva.nl" 24
declareCluster "fs4.das3.science.uva.nl" ".das3.science.uva.nl" 24

# Calculate nodes per cluster, and any remainder.
nodeRemainder=${1}
for i in `seq 1 $numberOfClusters`; do
        if [ ${maxNodes[${i}]} -ge ${nodeRemainder} ]; then
                nodesToReserve[${i}]=${nodeRemainder}
                nodeRemainder=0
                echo "Nodes to be reserved on ${fileServers[${i}]}:" \
                " ${nodesToReserve[${i}]}"
                echo "All processes assigned to $i clusters."
                break
```

```
        else
                nodesToReserve[${i}]=${maxNodes[${i}]}
                nodeRemainder=$((${nodeRemainder}-${maxNodes[${i}]}))
                multiClusterRun=$((${multiClusterRun}+1))
                echo "Nodes to be reserved on ${fileServers[${i}]}:" \
                " ${nodesToReserve[${i}]}"
        fi
done
if [ ${nodeRemainder} -ne 0 ]; then
        echo "There are not enough nodes available across all" \
             " clusters to run this job!"
        exit 1
fi

echo Number of processors to be reserved: $totalCPUs

# Reserve a number of nodes for a specified time, on the first cluster.
reserve=`preserve -1 -# ${nodesToReserve[1]} -t $reservationTime`
# Grep our reservation ID
reservationID=`echo $reserve | \
sed 's/Reservation number \([0-9]*\).*/\1/'`
echo Reservation ID = $reservationID

# Now we'll start waiting for the nodes to get assigned to us
echo -ne Waiting
# Store nodes in nodeList once the reservation is made
tempNodeList="-"
while [ "$tempNodeList" = "-" ]; do
  tempNodeList=`preserve -llist | grep $reservationID | \
  sed 's/.*[0-9][0-9]:[0-9][0-9]\t[a-z]*\t[\0-9]*\t\(.*\)/\1/'`
  echo -ne .
done
for i in $tempNodeList; do
  nodeList="$nodeList $i${suffices[1]}"
done

# Reserve all nodes on secondary clusters.
if [ ${multiClusterRun} -gt 1 ]; then
for i in `seq 2 $multiClusterRun`; do
  rsync -az --delete --force \
  $SCRATCH/hl_home ${fileServers[${i}]}:$SCRATCH
  rsync -az --delete --force \
  $SCRATCH/dat ${fileServers[${i}]}:$SCRATCH
  rsync -az --delete --force \
  $SCRATCH/hl_arc ${fileServers[${i}]}:$SCRATCH
  ssh ${fileServers[${i}]} "rm ~/profileData/*"
  tempNodeList=\
  `ssh ${fileServers[${i}]} \
  "source ~/.bash_profile && secondaryReserve.sh \
   ${nodesToReserve[${i}]} $reservationTime"`
  for j in $tempNodeList; do
    nodeList="$nodeList $j${suffices[${i}]}"
  done
done
fi

echo $'\n'$'\n'The Following nodes have been reserved \
and await further instructions: $nodeList

# Time to start the MPD's
for i in $nodeList;
do
        myriAddress=`ssh $i /sbin/ifconfig myri0 | grep \
        inet | sed 's/^.*inet addr://;s/ .*$//'`
        ssh $i "source ~/.bash_profile ; mpd $previousHost \
        --ncpus=$cpusPerNode --ifhn=$myriAddress --daemon \
        --listenport=$mpdListenPort"
# mpdexit localmpd > /dev/null
        sleep 0.5
        echo "Starting on $i: mpd $previousHost \
        --ncpus=$cpusPerNode --ifhn=$myriAddress \
        --daemon --listenport=$mpdListenPort"
#       previousHost="--host=$i --port=$mpdListenPort"
        previousHost="--host=$myriAddress --port=$mpdListenPort"
done;

# Set the correct start node (first node in the nodeList)
set $nodeList
startNode=$1
echo Starting from: $startNode
```

```
echo Starting MPD\'s successful
ssh -Y $startNode "source ~/.bash_profile && mpdtrace"

# Let's ROCK
echo Launching: $mpiRunCommand
echo Logfile is $logFile
ssh -Y $startNode "source ~/.bash_profile && $mpiRunCommand" > $logFile
echo DONE....

# We are done, shutdown all MPD daemons
ssh $startNode "source ~/.bash_profile && mpdallexit"

# Remove the reservations
# Rsync all the tau profiles from the secondary clusters
# to the primary cluster.
preserve -c $reservationID
if [ ${multiClusterRun} -gt 1 ]; then
for i in `seq 2 $multiClusterRun`; do
        ssh ${fileServers[${i}]} "source ~/.bash_profile && \
        secondaryReserveCancel.sh"
        rsync -az ${fileServers[${i}]}:$HOME/profileData $HOME
done
fi

# Restore original Env_system
cp $Env_system /var/scratch/$USER/hl_home/experiment/lib/Env_system
```

Hirlam is controlled by Mini-SMS, and it needs know how to submit a job. This is done in a file called `submission.db`. Since the compcentre variable declared in Env_System is changed to DAS3, we now have to tell Mini-SMS what to do in case of DAS-3. A template submission.db can be checked out using the following command:
`cd ~/hl_home/experiment && Hirlam co scripts/submission.db`
However a customized version suited for DAS-3 can be used instead of adapting the template, see listing 16.

Listing 16: submission.db

```
# $Id: submission.db 5783 2008-03-18 15:17:19Z ovignes $
# define the  jobs and their submission sequences
#-------------------------------------------------

# Gerard Cats, July 2004

$instructions = <<EOI;
This file is the data file to be used by "Submit.pl".
A job file will be constructed and this job file will be submitted
with the string $submit (in which the title of the job file to be
submitted must be $jobfile-q).
Both the job file title and $submit will be edited according to
the hash in %edit (strings of the form %$key% will be replaced by
$value. This currently will not be done recursively).
The constructed job file will look like:

headers
input job file (with substituted export variables)
trailers

headers and trailers are given below. They are also edited according
to %edit.
headers and trailers are constructed by a subroutine because
they are multi-line and perhaps multi-complex.

You are allowed to change the environment variables. E.g. if COFLAGS
is exported in the input job file by a line exactly matching:
COFLAGS="-d 2004/12/31" export COFLAGS
then assigning $COFLAGS = "\"-d 2004/12/30\""  will result in the line
COFLAGS="-d 2004/12/30" export COFLAGS
in the $jobfile-q file. Note that you have to explicitly set the
surrounding quotes by including quoted quotes (\") when you set
$COFLAGS. This, of course, is needed for all environment variables
of which the value contains shell metacharacters.

First a set of defaults for $submit, %edit, etc. will be given,
then overrides per job are allowed. For this, jobs are identified by
$SMSNAME.
```

```
EOI

# the following jobs can be skipped

# Task Execute with empty SCRIPT:
if ( $SMSNAME =~ m~/Execute$~ && $SCRIPT =~ m~/^\s*$~ ) {
   $complete = 1; exit;
}

# replace a directory structure by a long title,
# separated by % - signs:
$smsname = $SMSNAME; $smsname =~ s~/~%~g;

# information about ensemble mode
$ENSSIZE = $ENV{ENSSIZE};
$ENSSIZE = -1 if (not defined $ENSSIZE);
$ENSSIZE = sprintf "%d", $ENSSIZE; # assure decimal numeric

# defaults:

$host = 0;      # run on this host ($host is the host number)

# The following jobs are not submitted through (mini-)SMS:
if ( $SMSNAME =~ m~BoundariesChild~ ) {
   $__mSMS__ = 0;
   $__SMS__  = 0;
}

# default jobout title
unless ( defined $jobout ) {
   $jobout = "$smsname.$SMSTRYNO";
}

# For LSMIX runs, skip reruns at intermediate hours
# and on first cycle if started with RUN=1
if ( $ENV{LSMIX} eq 'yes' ) {
   my $dtg = sprintf("%08d%02d",$YMD,$HH);
   #intermediate hour
   $complete = 1 if ($RUN > 1 and ($HH % $ENV{MIXINT}));
   $complete = 1 if ($dtg eq $ENV{DTGBEG}
                     and $RUN > $ENV{StartRUN});
   $complete = 1 if ($dtg ne $ENV{DTGBEG}
                     and $RUN > 1 and $SCRIPT eq 'VARan');
   exit if $complete;
}

# override defaults at DAS3

if ( $COMPCENTRE eq "DAS3" ) {

# no COMPCENTRE given: run everything in the background
} else {
   $bckgrd = 1;
}

# The number of tasks GNU-make is allowed to spawn in parallel:
   $nparallel_make = 8;
   $JMAKE="-j$nparallel_make";

# several jobs have deviating properties:

if ( $COMPCENTRE eq "DAS3" ) {
        $SMSHOST = $SMSNODE;
        $host= 0;

        # NSLOTS equals the number of computational nodes
        # NTASKS equals the number of tasks per node

        #Default 1 slot with 1 cpu;
        $NSLOTS   = 1;
        $NTASKS   = 1;
        $NPROC_HGS = 0;

        #Default nprocx, nprocy, nproc_hgs;
        $nprocx    = 1;
        $nprocy    = 1;
        $nproc_hgs = 0;

        # Standard job submit, running on 1 cpu on 1 node
```

```
        $submit = "reserve.sh $NSLOTS $NTASKS $NPROC_HGS \
$Env_system $jobfile-q $jobout &";
   if ($SMSNAME =~ m~/PpAn/Execute~ and $SCRIPT =~ m~Postpp~)
   {
      $NSLOTS = $ENV{NODES};
      $NTASKS = $ENV{TASKS_PER_NODE};
      $NPROC_HGS=0;

      $nprocx    = $ENV{NPROCX}    || $nprocx;
      $nprocy    = $ENV{NPROCY}    || $nprocy;
      $nproc_hgs = $ENV{NPROC_HGS} || $nproc_hgs;

      $submit = "reserve.sh $NSLOTS $NTASKS $NPROC_HGS \
$Env_system $jobfile-q $jobout &";

      $jobclass = "parallel";
      #Mark procx * nprocy must match nr of compute processors
      $nprocxy = 1;
   }
   elsif ($SMSNAME =~ m~/Forecast/Execute~ and $SCRIPT =~ m~Prog~)
   {
      $NSLOTS = $ENV{NODES};
      $NTASKS = $ENV{TASKS_PER_NODE};
      $NPROC_HGS= $ENV{NPROC_HGS};

      $nprocx    = $ENV{NPROCX}    || $nprocx;
      $nprocy    = $ENV{NPROCY}    || $nprocy;
      $nproc_hgs = $ENV{NPROC_HGS} || $nproc_hgs;

      $submit = "reserve.sh $NSLOTS $NTASKS $NPROC_HGS \
$Env_system $jobfile-q $jobout &";

      $jobclass = "parallel";
      # mark procx * nprocy must match nr of compute processors
      $nprocxy = 1;
   }

   if (defined($nprocxy))
   {
      unless ($nprocx * $nprocy + $nproc_hgs ==
      $ENV{ NODES } * $ENV{ TASKS_PER_NODE} )
      {
         print STDERR "NPROCX * NPROCY + NPROC_HGS is not equal to" .
                      " NODES * TASKS_PER_NODE:\n";

         print STDERR "$nprocx * $nprocy + $nproc_hgs !=" .
                      "$ENV{NODES} * $ENV{TASKS_PER_NODE}\n";
         exit 1;
      }
   }

} else {
# default nprocx, nprocy, nproc_hgs:
   $nprocx      = 1;
   $nprocy      = 1;
   $nproc_hgs   = 0;

   if ( $SMSNAME =~ m~/PpAn/Execute~ and $SCRIPT =~ m~Postpp~ ) {
      $nprocx    = $ENV{ NPROCX }    || $nprocx;
      $nprocy    = $ENV{ NPROCY }    || $nprocy;
      $nproc_hgs = $ENV{ NPROC_HGS } || $nproc_hgs;
   } elsif ( $SMSNAME =~ m~/Forecast/Execute~ and
             $SCRIPT =~ m~Prog~ ) {
      $nprocx    = $ENV{ NPROCX }    || $nprocx;
      $nprocy    = $ENV{ NPROCY }    || $nprocy;
      $nproc_hgs = $ENV{ NPROC_HGS } || $nproc_hgs;
   } elsif ( $SMSNAME =~ m~CheckOptions/Execute~ and
             $SCRIPT =~ m~Checkoptions~ ) {
      $nprocx    = $ENV{ NPROCX }    || $nprocx;
      $nprocy    = $ENV{ NPROCY }    || $nprocy;
      $nproc_hgs = $ENV{ NPROC_HGS } || $nproc_hgs;
   } elsif ( $SMSNAME =~ m~/FCinput/Execute~ and
             $SCRIPT =~ m~FCinput~ ) {
      $nprocx    = $ENV{ NPROCX }    || $nprocx;
      $nprocy    = $ENV{ NPROCY }    || $nprocy;
      $nproc_hgs = $ENV{ NPROC_HGS } || $nproc_hgs;
   } elsif ( $SMSNAME =~ m~/AnUA/Execute~ and
             $SCRIPT =~ m~PertAna~ ) {
      $nprocx    = $ENV{ NPROCX }    || $nprocx;
      $nprocy    = $ENV{ NPROCY }    || $nprocy;
```

```
      $nproc_hgs = $ENV{ NPROC_HGS } || $nproc_hgs;
   } elsif ( $SMSNAME =~ m~/Boundaries~ ) {
      $edit{ LL_MEMORY_LIMIT }          = "1500mb";
   } elsif ( $SMSNAME =~ m~/AnSFC~ ) {
      $edit{ LL_MEMORY_LIMIT }          = "2500mb";
   } elsif ( $SCRIPT =~ m~Climate~ ) {
      $edit{ LL_MEMORY_LIMIT }          = "1500mb";
   }
}
# InitRun and LogProgress must be run on the local host,
# in the background
# A number of extra processes qualify for this as well.
if ( $SMSNAME =~ m~InitRun|MakeVerif|LogProgress|CheckOptions/Execute|
                  BuildOthers/Build|BuildClimate/Build~ ) {
   $bckgrd = 1;
}

if ( $bckgrd ) {
   $SMSHOST     = $SMSNODE;
   $submit = "$jobfile-q > $jobout 2>&1 &";
   $host = 0;
}

# create jobout directories
   @joboutdir=split('/', $jobout);
# last element in path is the filename, not a directory
   $#joboutdir -= 1;
   foreach ( @joboutdir ) {
      $j_h .= $_ . '/';
      unless ( -d $j_h ) {
         undef $!;
         mkdir $j_h, 0755;
         if ( $! )
          {print STDERR "Could not mkdir $j_h: $!\n"; exit 1}
      }
   }

# Env_system and PseudoEnvironment for this host
$Env_system = ${"HL_LIB$host"} . "/Env_system";
$SETENV = ${"HL_LIB$host"} . "/config-sh/PseudoEnvironment";

# create headers

$headers        = &Headers ( $SMSNAME );

## Don't queue this job if it is a boundary interpolation job
## that has nothing to do
if ( $SMSNAME =~ m~/LBCn/LBC(\d+)/~ ) {
   $bdint = $ENV{BDINT} || 6;
   $bdll = $1 * $bdint;
   $bdlast = int(($LL + $bdint - 1)/$bdint) * $bdint;
   if ( $bdll > $bdlast ) {
      $complete = 1;
      exit;
   }
}

# create trailers

$trailers       = &Trailers( $SMSNAME );

1;      # to succeed do file

# ---------------------------------------
sub Headers{
# Headers: template for headers
# synopsis: $jobheaders = Headers( $JOB )
# author: Gerard Cats, 15 April 2004
   my $job = shift;
   my $HH=sprintf "%2.2d", $HH;
   my $lines = "";
   $lines .= <<EOH;
#!/bin/sh
EOH

# include the mini-SMS headers
   if ( $__mSMS__ ) {
      if ( $SMSHOST eq $SMSNODE ) {     # runs on the local node
         $lines .= <<EOH;
smsinit()
```

```
{ echo > $SMSHOME/$smsname.active; }
smsabort()
{ echo > $SMSHOME/$smsname.aborted;  echo 'SMS-> aborted';  }
smscomplete()
{ echo > $SMSHOME/$smsname.complete; echo 'SMS-> complete'; }
EOH
      # on a remote host: rcp to SMSNODE:SMSHOME
      } else {
      $lines .= <<EOH;
smsinit()
{ echo > $smsname.active;   $RCP $smsname.active \
         $SMSNODE:$SMSHOME; rm -f $smsname.active;   }
smsabort()
{ echo > $smsname.aborted;  $RCP $smsname.aborted \
         $SMSNODE:$SMSHOME; rm -f $smsname.aborted
  echo 'SMS-> aborted'; }
smscomplete()
{ echo > $smsname.complete; $RCP $smsname.complete \
         $SMSNODE:$SMSHOME; rm -f $smsname.complete
  echo 'SMS-> complete'; }
EOH
      }
   }
   $lines .= <<EOH;
ulimit -S -s unlimited || ulimit -s
ulimit -S -m unlimited || ulimit -m
ulimit -S -d unlimited || ulimit -d

ulimit -a
EOH

# a sample algorithm (how to vary LL with time of day:
# long forecast at main hours only)
# note that still LL may be overwritten by a
# command-line argument

   $llmain = $ENV{LLMAIN} || $ENV{LL} || '48';
   if ( $ENV{FGAT} eq 'yes' ) {
      $llshort = sprintf("%02d", $ENV{FCINT}+int($ENV{FCINT}/2));
   } else {
      $llshort = sprintf("%02d", $ENV{FCINT});
   }
   if ( $ENV{ANALYSIS} eq '3DVAR' and $ENV{FCINT} < 06 ) {
      $llshort = 06;
   }
   if ( $ENV{LSMIX} eq 'yes' and $RUN > 1 ) {
      if ( $ENV{ANALYSIS} eq '4DVAR' && $ENV{$DTG} ne $ENV{$DTGBEG} ) {
         $obscut = $ENV{OBSCUT} || '0130';
         $LL = sprintf("%02d", int($obscut/100 + 1));
      } else {
         $LL = $llshort;
      }
   } else {
      if ( $HH % 12 ) { $LL = $llshort; } else { $LL = $llmain; }
   }
   if ( int $ENV{ENSMBR} >= 0 ) {
     if ($ENV{ENSDA} =~ /:$ENV{ENSMBR}:/ &&
         $ENV{EnsCycle} !~ /:$HH:/) {
       $LL = $ENV{ENSDALL};
     }
   }

   # experiment ref has LL=06 by default
   if ( $EXP eq 'ref' ) { $LL = '06' }
   $lines .= "LL=$LL export LL\n";

# set NPROCX, NPROCY, NPROC_HGS
   $lines .= "NPROCX=$nprocx export NPROCX\n";
   $lines .= "NPROCY=$nprocy export NPROCY\n";
   $lines .= "NPROC_HGS=$nproc_hgs export NPROC_HGS\n";

   return $lines;
}
# ---------------------------------------
sub Trailers{
# Trailers: template for trailers
# synopsis: $jobheaders = Trailers( $JOB )
# author: Gerard Cats, 15 April 2004
   my $job = shift;
   my $lines = "";
```

```
# code to wait for completion
# but under (mini-)SMS, (mini-)SMS will do it
   unless ( $__mSMS__ or $__SMS__ ) {
      if ( $LOADLEVELER eq 'yes' ) {
         # LoadLeveler ---------------------
         # LoadLeveler needs a list in JOBID
         $lines .= <<\EOH;
MAXWAIT=${MAXWAIT-10800}
for job in $JOBID; do
   maxwait=0
   [ $maxwait -ge $MAXWAIT ] && { llcancel $job; continue; }

   while [ 1 ]; do
      llq "$job" -r %%st %%c >job_ll || { llcancel $job; break; }
      [ `egrep -c "\!" job_ll` -le 0 ] && break
      sleep 10
      maxwait=`expr $maxwait + 10`
      [ $maxwait -ge $MAXWAIT ] && { break; }
   done

   rm -f job_ll
done
EOH
      } else {          # non-LoadLeveler: wait for the background jobs
         $lines .= <<EOH;
wait
EOH
}

   } else {     # (mini-)SMS code
      $lines .= <<EOH;
smscomplete
trap - 0
EOH
   }
   return $lines;
}
```

In the latest version of Hirlam, during the verification phase, some data preparation for the Harmonie weather forecasting model is done. Due to a bug in the makefile, this part of the verify script will cause Hirlam to fail. This can be solved two ways: the first is patching the makefile, the second is to disable this part of the verification phase. There is no need to have any Harmonie related data on the system, so the latter solution should do fine. Instructions to bypass this verification are shown in listing 17.

Listing 17: Verify phase patch

```
#Check out the Env_expdesc script from the Hirlam repository
#to the experiment directory with:

Hirlam co scripts/Env_expdesc

#Edit this script and replace the line
#VERIFY:ve:vf: by the line VERIFY=:ve:
```

Without proper input data there can be no experiment, therefore the input data should be copied to the appropriate directories, a special script is created to automate this. The script prepares the following directories:

- `/var/scratch/$user/hl_arc`
- `/var/scratc/$user/hl_dat`

The original template directories reside in the experiment directory[15]. The script responsible for copying this data is also used to start the Hirlam experiment itself, the script can be found in listing 18.

[15]They should be copied there from the accompanying CD

Listing 18: Script to start Hirlam experiment

```
#!/bin/sh
source ~/.bash_profile
SCRATCH=/var/scratch/$USER
if [ -d $SCRATCH/hl_arc ]
then
        echo "Experiment already has a hl_arc directory,"
              "not copying premade_hl_arc..."
else
        echo "Starting a new experiment, copying over premade_hl_arc."
        cp -R premade_hl_arc $SCRATCH
        mv $SCRATCH/premade_hl_arc $SCRATCH/hl_arc
fi
if [ -d $SCRATCH/dat ]
then
        echo "Data directory already exists."
else
        echo "Data directory does not exist, copying over premade_dat."
        cp -R premade_dat $SCRATCH
        mv $SCRATCH/premade_dat $SCRATCH/dat
fi
Hirlam start DTG=2007011600 DTGEND=2007011600 LLMAIN=12
```

Make sure the ssh session to the file-server was established using X forwarding, otherwise Hirlam won't be able to supply visual progress information on the forecast through Mini-SMS.

A.5.3 Multicluster

To run Hirlam on multiple clusters, the multicluster reservation script in listing 15 must be used in the user's experiment directory. In your `hl_home/experiment/scripts` directory, remove the `reserve.sh` symbolic link and make a new one with:
`ln -s reserve-multicluster.sh reserve.sh`
Edit `reserve.sh` and change the commands in listing 19 to control how many nodes the reservation script may use for each cluster.

Listing 19: Editing the nodes available per cluster

```
# Edit k, l, n, m to control assignment of processes on each cluster.
declareCluster "fs1.das3.liacs.nl" ".das3.liacs.nl" k
declareCluster "fs0.das3.cs.vu.nl" ".das3.cs.vu.nl" l
declareCluster "fs2.das3.science.uva.nl" ".das3.science.uva.nl" n
declareCluster "fs4.das3.science.uva.nl" ".das3.science.uva.nl" m
```

Furthermore, there are some additional caveats that must be dealt with.

The Pathscale compiler is only present on the LIACS location. An application compiled by Pathscale uses dynamic linking against a Pathscale library, which cannot be found on other clusters. To fix this, one should copy `/usr/local/pathscale` to `~/local/pathscale`, add `$HOME/local/pathscale/lib/3.0` to `LD_RUN_PATH` and add `$HOME/local/pathscale/bin` to `PATH`, as well as update the pathscale library paths in config.linuxgfortran and the MPICH2-MX engage script. After this is done, MPICH2-MX and Hirlam need to be recompiled.

MPICH2-MX requires public/private key authentication to the other clusters for a multicluster run. Add your ssh key from `~/.ssh/id_dsa.pub` to the `~/.ssh/authorized_keys` file of all other clusters.

The `~/local` directory as well as your `~/.bashrc` and `~/.mpd.conf` files must be synchronized to all other clusters so the necessary environment variables and libraries are present for the multicluster run. Do this with the commands in listing 20, for each of the fileservers of the secondary clusters.

Listing 20: Synchronizing your home directory with other clusters

```
# Replace fs0.das3.cs.vu.nl with the other fileservers you
   need to sync to.
cd
rsync -az --delete --force local fs0.das3.cs.vu.nl:
scp .bashrc fs0.das3.cs.vu.nl:
scp .mpd.conf fs1.das3.cs.vu.nl:
```

Hirlam will crash if the HGS processes of the forecast do not run on the primary cluster. Edit `Env_system` and uncomment the following line:
`export HGS_IO_RANKS="2 3"`
This will run the HGS processes on ids 2 and 3[16], which be run on the primary cluster as long as there are at least 2 nodes assigned to the primary cluster in the reservation script.

If the number of nodes used for a Hirlam run is less than half of the total nodes available to the cluster's reservation system, Mini-SMS will attempt to start the Analysis Postprocessing and the Forecast steps at the same time. The multicluster reserve script is not designed to handle simultaneous MPI runs, therefore Hirlam will crash without an apparent explanation. In your experiment directory, check out the Mini-SMS task description file with `Hirlam co scripts/hirlam.tdf` and edit the file according to listing 21. This will make the Forecast dependent on the Postprocessing step, so Mini-SMS will no longer try to launch these steps simultaneously.

Listing 21: Making the Forecast depend on the Postprocessing step

```
# Around line 626, replace this line:
trigger ( ( AnUA/Listen2AnUA == complete ) && \
# by this line:
trigger ( ( PpAn/Listen2PpAn == complete ) && \
```

[16] Add more ids in ascending order if you need more HGS processes.

Appendix B TAU Profiling

This appendix will describe how to compile and link TAU against Hirlam.

Download the TAU sources from the following URL:
`http://www.cs.uoregon.edu/research/tau/downloads.php`
You will have to enter your name, email address and a small comment. Place the source tarball in `~/src` and extract the source. Compile and install TAU with the script in listing 22.

Listing 22: TAU Engage script

```
#!/bin/sh

#
# TAU - MPI Profiling Tool Configure and install script for DAS-3
# location Liacs (Leiden University) by Cor Cornelisse
#
# This script assumes AMD64 Opteron architecture and
# mpich2 + pathscale software present. Set the correct
# paths below.
#

mpich2=${HOME}/local/mpich2

libdir_mpich2=${mpich2}/lib
includedir_mpich2=${mpich2}/include

export LD_LIBRARY_PATH=${libdir_mpich2}

installdir=$HOME/local/tau
mkdir -p $installdir

./configure -prefix=$installdir \
-arch=x86_64 \
-c++=pathCC \
-cc=pathcc \
-fortran=pathscale \
-mpiinc=${includedir_mpich2} \
-mpilib=${libdir_mpich2} \
-mpilibrary="-lmpich -lmpichcxx -lmpichf90" > logconfig 2>&1

make > logmake 2>&1

make shared > logmakeshared 2>&1

make install > logmakeinstall 2>&1

mv logconfig logmake logmakeshared logmakeinstall $installdir/
cp $0 $installdir/
```

In `~/.bashrc`, add
`$HOME/local/tau/x86_64/bin` to `PATH`, and add
`$HOME/local/tau/x86_64/lib` to `LD_RUN_PATH`. TAU needs to know where to store profiled data, so also add the following line to `~/.bashrc`:
`export PROFILEDIR="$HOME/profileData"`. Of course this directory needs to be present, so execute `mkdir ~/profileData` to create it.

The necessary variables in the Hirlam `config.linuxgfortran` for the TAU libraries and libdirs have already been set. However the TAU libraries need to actually be linked to hirlam now. Edit the file as shown in listing 23.

Listing 23: Adding TAU libraries to config.linuxgfortran

```
# The following 3 lines are to be replaced:
LDFLAGS_VAR := $(LDFLAGS) $(MPILIBS) $(LAPACKLIBS) $(FFTLIBS
    )
LDFLAGS_CMA := $(LDFLAGS) $(MPILIBS) $(LAPACKLIBS)
LDFLAGS_HIR := $(LDFLAGS) $(MPILIBS) $(LAPACKLIBS) $(FFTLIBS
    )
```

```
# At line 96, replace the above 3 lines with the following:
LDFLAGS_VAR := $(LDFLAGS) $(TAULIBS) $(MPILIBS) $(LAPACKLIBS
    ) $(FFTLIBS)
LDFLAGS_CMA := $(LDFLAGS) $(TAULIBS) $(MPILIBS) $(LAPACKLIBS
    )
LDFLAGS_HIR := $(LDFLAGS) $(TAULIBS) $(MPILIBS) $(LAPACKLIBS
    ) $(FFTLIBS)
```

Recompile hirlam with the following command:
`cd ~/src/hirlam/current ; make clean && make`
Now your TAU profiles will appear in the `~/profileData` directory after a forecast. The included TAU program Paraprof can be useful in examining the profile data. With the TAU `bin` directory in your `$PATH`, Paraprof can be launched on the file server from a terminal with X forwarding enabled.