Development of two Science Investigator-led Processing Systems (SIPS) for NASA's Earth Observation System (EOS)

Curt Tilmes, GSFC
NASA Goddard Space Flight Center
Greenbelt, MD 20771
Email: Curt.Tilmes@nasa.gov

Mike Linda, SAIC
7501 Forbes Blvd., Suite 103
Seabrook, MD 20706
Email: Mike.Linda@gsfc.nasa.gov

Albert J. Fleig, PITA Analytic Sciences
8705 Burning Tree Rd.
Bethesda, MD 20817
Email: Albert.J.Fleig@gsfc.nasa.gov

Abstract— **While building a series of large data processors for remotely sensed data, GSFC LTP generalized the approach and developed a universal system architecture. With a suite of plug-in modules for a variety of functions, the design is a customizable data processing system driven by a rich set of automated production rules. It can take high volumes of diverse inputs, recognize data set types, run many separate processes simultaneously as well as sequentially against the data, and automatically send resulting products to end users. Implemented with commodity hardware and open source software, the highly scalable system has been proven in a number of applications ranging in size from tiny to huge.**

I. Introduction

NASA Goddard Space Flight Center (GSFC) Laboratory for Terrestrial Physics (LTP) has been building data processing systems for a number of years. We have now evolved several systems. Most recently, starting in 2001, the LTP developed a Science Investigator-led Processing System (SIPS) for the Ozone Monitoring Instrument (OMI). Based on experience from building previous systems, we generalized the approach and showed that it can be quickly customized to a number of missions. More importantly, due to the nature of the science work that we support, our approach and the system architecture play together in an uncommon way where hardware is acquired as the project evolves. Our approach presents several advantages including much reduced cost and a resilient system that is highly adaptable and scalable.

II. Background

OMI, to be launched on the Aura spacecraft in mid 2004, is a European contribution to the Earth Observing System (EOS). OMI measurements will be highly synergistic with data from other instruments on the EOS Aura platform. OMI will provide daily global coverage. It will continue the Total Ozone Mapping Spectrometer (TOMS) record of ozone and related atmospheric chemistry parameters.

The Netherlands's Agency for Aerospace Programs (NIVR), in collaboration with the Finnish Meteorological Institute (FMI) and the Royal Netherlands Meteorological Institute (KNMI), sponsored OMI construction. The OMI science team, led by a principal investigator from the Netherlands, includes government and industry participants from the Netherlands, Finland, and the United States. The team is developing algorithms, calibration, data processing, quality assessment, validation, and analysis software. Much of this science software is being hosted on the LTP's OMI SIPS.

III. Generalized SIPS

A SIPS can be simplified to a context diagram shown in Fig. 1. Data providers, such as remote sensing instruments, or national weather centers, supply data. The data is ingested, processed, and exported to receivers. The processing system includes a way for human operators to control it and an interface for the curious to see what is going on.

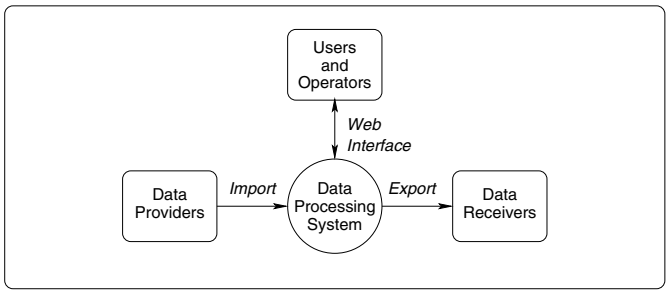

Fig. 1. Data Processing Context Diagram

Our approach to a generalized SIPS considers that the data processing can be split into combinations of simultaneous and sequential production steps. Although we sometimes call the simultaneous execution of science code "parallel processing," the missions we support need to be distinguished from massively parallel processing projects—we are not one of those. The remote sensing algorithms that we deal with are developed to execute independently of each other. Some feed data sequentially from one process into another, but many have no connection with others except for occasionally using the same input files. In response, we developed a general purpose SIPS that drives science processes simultaneously and in sequences.

Satellite-acquired data often comes in discrete chunks that can be dealt with independently of each other. Processing

0-7803-8742-2/04/$20.00 (c) 2004 IEEE

many independent chunks at the same time permits an increase in performance of the overall system. In designing a SIPS, we took advantage of such natural separation of data and algorithms.

Satellite instrument data is typically processed in levels. Level zero (L0) is raw data. Level 1B (L1B) is calibrated and geolocated pixel-level radiance. Level two (L2) is also pixel-level data, but converted to physical quantities (ozone, water vapor, temperature, etc.). Level three (L3) is data re-sampled in time or space; it is often aggregated from several L2 files.

Most of our SIPS processing is separated by processing level and by products within processing levels. For example, L0 data from MODIS and OMI comes in 2-hour sets. The MODIS SIPS processes L0 data into 5-minute L1B granules and the OMI SIPS generates one-orbit (90 minute) L1B files. L2 processing in both systems maintains the same granule size as their L1B. Since processing one data granule through one algorithm is often independent of processing the same granule through another algorithm, two algorithms can run simultaneously against the same input files. Similarly, processing one granule through one algorithm, and processing the next granule through the same algorithm is also often independent of each other. L3 data can be often generated in independent spatial regions that are smaller than the global data set. A SIPS can take advantage of such independent processing and run many steps simultaneously.

Over the years we evolved a generalized SIPS architecture that is shown in Fig. 2. It can drive science software in arbitrary and complex combinations of simultaneous and sequential processing steps.

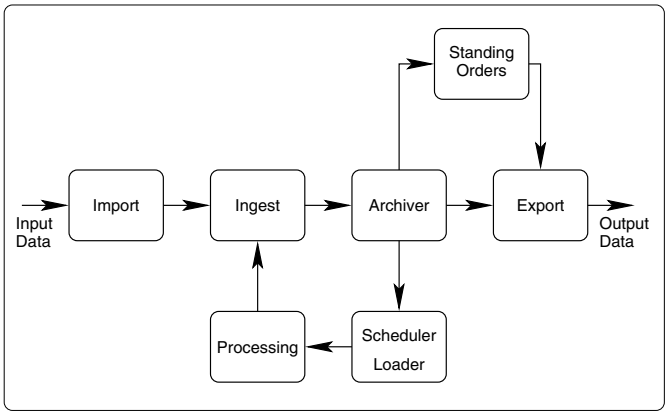

Fig. 2. SIPS System Architecture

The major functional components of the generalized SIPS were designed to be loosely coupled. The functions can be hosted on separate machines as discussed in a later section on scaling. Data within the system is handed off from one discrete function to the next.

Data flow begins with an Import function as shown in Fig. 2. Import provides a storage area where external entities can deposit files. Import encapsulate handshaking that is unique to each external entity. When new input files become available, Import transports the files in and hands them to Ingest.

Ingest parses metadata associated with inputs and determines the type of received files. It hands files and information about the files to the Archiver.

The Archiver manages a storage system. It saves files and keeps track of them with entries in a small database.

The Scheduler checks for new file entries in the archive and compares them against production plans using production rules as a guide. Production plans are high level definitions of what should get processed. Production rules are detailed definitions for each science algorithm that describe what combinations of input files are needed for each execution of the science software. The Scheduler starts with production plans that it expands into details, then it interprets the details using the production rules. It then matches the interpretations against input file availability. The result is a decision whether or not there are enough input files available in the archive to initiate a particular algorithm execution.

Let's follow an example to illustrate how the Scheduler works. Suppose that a production plan calls for the Ozone algorithm to be executed for the month of June. When a human operator enables the plan, the Scheduler expands it using the Ozone algorithm's production rules. The rules might say that the algorithm should run once per 90-minute orbit. The month of June has 30 days and therefore there will be approximately (30 days) x (24 hours/day) x (60 minutes/hour) x (1 execution/90 minutes) = 480 executions of the algorithm for June. The Scheduler then determines a list of discrete input files for each execution. The production rules might say the the Ozone algorithm needs L1B radiances, so the Scheduler might determine that it needs an L1B file of radiances that spans from May 31 23:50 to June 1 01:20 for one of the executions. In addition, the production rules might also call for an L1B solar irradiance file that is generated (for OMI) once per day. The Scheduler would look for an irradiance file that temporally overlaps the L1B radiance file. With such a list of needed files, the Scheduler queries the database to determine which files in the archive match the criteria. If the query shows that all files are available, the Scheduler initiates the algorithm execution. If the files are not all available, the Scheduler tries the same set of queries a while later. The Scheduler repeats the process until all executions in the enabled plan have been initiated.

When an algorithm execution is initiated, the Loader gets the file list from the Scheduler and retrieves the needed files from the archive. It loads the files into a processing node. Processing then initiates the science algorithm against the retrieved files.

When Processing finishes, it transmits the output files to Ingest. Output files are new files for the archive, and just as with any new files, Ingest parses metadata associated with the files, determines file types, and hands the files to the Archiver.

Once files appear in the archive, the Standing Orders processor matches up new files with user orders. For matches, the Standing Orders processor uses Export to retrieve copies of the files from the archive and send them out to end users.

IV. MODAPS

The LTP developed a SIPS for the Moderate Resolution Imaging Spectrometer (MODIS). The MODIS Data Processing System (MODAPS) has been in full operation since the launches of the Terra and Aqua spacecrafts in December 1999 and May 2002 respectively.

Since its early stages, MODAPS has continually evolved to better support the MODIS mission. MODAPS originated out of a 1997-vintage SeaWiFS Data Processing System. Initially, MODAPS was the MODIS Emergency Backup System (MEBS) that later became "version zero" (V0) MODAPS. By the time of Terra's launch, MODAPS evolved to version one (V1) described earlier [1]. With the launch of Aqua, MODAPS evolved again into version two (V2) in which the software architecture was somewhat generalized and the software implementation was streamlined and made more robust. MODAPS version three (V3) came about a year later. It included a hardware topology shown in Fig. 3 in which the central supercomputer is augmented with 2-processor Linux servers connected by a network. The result is an "asymmetric cluster" [2].

We now run multiple instances of MODAPS. The system can process and reprocess MODIS data at a rate many times faster than real time.

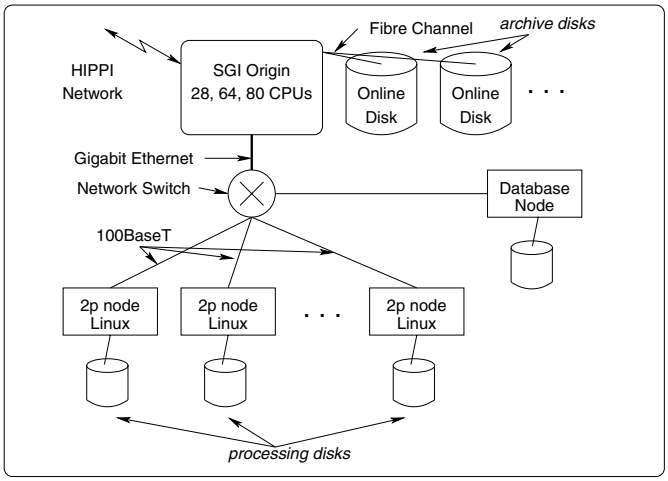

Fig. 3. MODAPS V3 Hardware Architecture

V. FROM MODAPS TO OMIDAPS

MODAPS started out constrained by the original EOS heritage. The EOS system design called for centralized massive computers with shared memory. The idea was to process immense amounts of data all at once. SGI multi-CPU mainframes were selected for the EOSDIS.

Our first MODIS SIPS followed the EOS approach—until we realized that we had functions we did not need. We were hosting largely independent processes on CPUs that did not have to communicate closely with each other. Memory did not need to be shared between CPUs. The storage system did not need very high speed I/O between CPU and disk for simultaneous shared access to large data sets. Unique capabilities of the large SGI mainframe were not key for MODIS. We realized we could buy the same computing capability, without special mainframe features, at a lower cost.

As we added Intel Linux servers into MODAPS V3, and had a chance to load the small Linux machines with lots of work, we found that inexpensive computers were very capable and robust. Since the job of the mainframe was actually a set of several jobs that could be hosted on separate machines, the multi-CPU mainframe could be replaced by loosely coupled smaller machines. With sufficient communication between the units, all functions of a processing system could be split up into individual computers connected by a network. Also bulk archiving could migrate from the MODAPS single large file server to a farm of smaller file servers. Since technology has now reached a point of providing sufficient networking, a large cluster of commodity processors and disks could be used for constructing the entire SIPS.

In 2001, when we set out to evolve MODAPS into the next generation processing system, OMIDAPS, we considered the differences and similarities between the two. Although the science between MODIS and OMI is quite different, running large numbers of science algorithms systematically and repeatedly against vast quantities of data remains the same. Both systems have essentially identical non-science system requirements. As a result, MODAPS and OMIDAPS system architectures are identical at the level shown in Fig. 2. But instead of simply replicating a MODAPS, we decided to evolve the system so it can be hosted on less expensive hardware.

At the same time that we switched to commodity computers, we also changed to open source software. The operating system became Linux, and the database was implemented using PostgreSQL. Having a system that is based completely on open source allows the entire implementation to be distributed and used without license concerns or costs.

One other notable change between MODAPS and OMIDAPS includes a generalized graphical user interface and reporting structure. While redesigning it, we also reimplemented it as a web-based HTML system instead of the MODAPS Java-based GUI. The web interface is hosted on an Apache web server and implemented mostly using scripts written in Perl and Mason.

VI. OMIDAPS

The OMI Data Processing System (OMIDAPS) hardware architecture is shown in Fig. 4. A central network switch connects a number of small servers that play various roles in the system. The functions shown in Fig. 2 are typically spread across one or more machines. We will say more about that in the subsection on scaling.

Fig. 5 depicts the major interfaces between OMIDAPS and external entities. OMI sends raw L0 data to ground stations. It then flows through several NASA systems that are external to OMIDAPS. Within minutes of acquisition by the spacecraft,

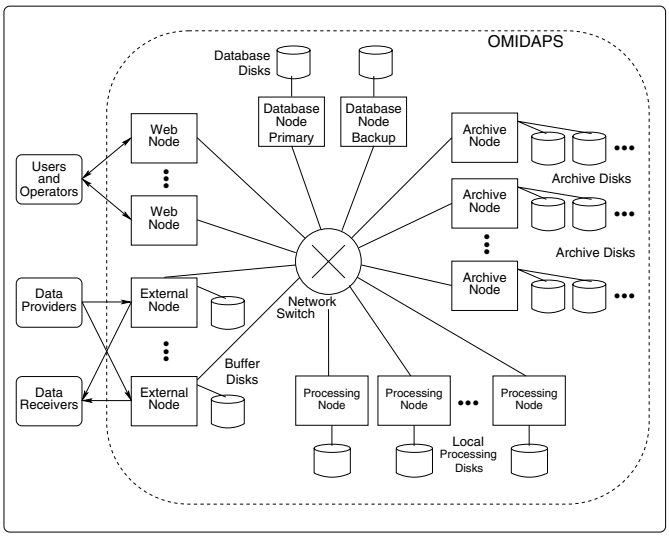

Fig. 4. OMIDAPS Hardware Architecture

the L0 data will be stored in the Goddard Earth Sciences Distributed Active Archive Center (GES-DAAC).

OMIDAPS receives OMI L0 data from the GES-DAAC through a buffer server. Once in OMIDAPS, the L0 data is processed into L1B products. Calibrated and geolocated L1B is then fed to science algorithms that produce L2 and L3 outputs. All standard data products, L1B through L3, are transmitted back to the GES-DAAC for long term archive and distribution to the world.

OMIDAPS also provides direct data distribution to the OMI Science Team for a number of purposes including quality assessment, algorithm improvement, calibration, and trending.

We have several instances of OMIDAPS. A Development instance is used for integration and testing of new OMIDAPS features as well as for unit testing of OMI science processes as they get delivered from the Science Team. A larger Test instance is used to ensure operational readiness of the integrated system. The Test instance is functionally identical to the Production instance (whereas the development instance may not always be identical when new OMIDAPS features are created). The Test instance is also used for science software characterization that is key for cluster tuning; cluster tuning is discussed later. Tested algorithms are then installed into a Production instance that processes real OMI data in bulk. Although different in size, all the OMIDAPS instances are functionally identical. That is key for development and testing: there is no sense proving that something works when it is different from the target.

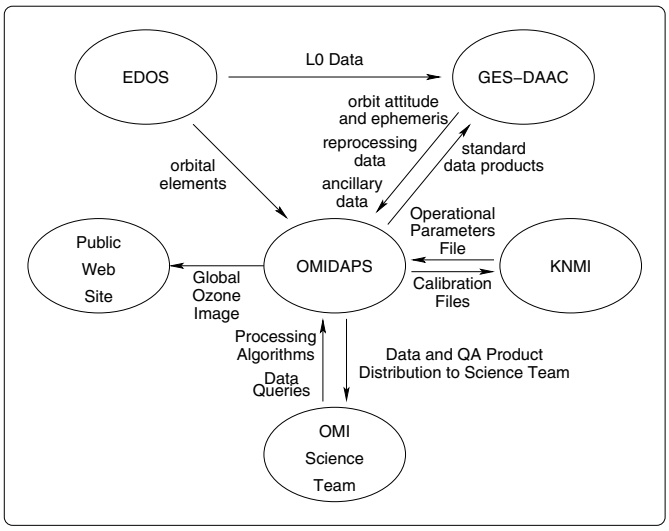

Fig. 5. OMIDAPS Interfaces

A. Scaling

Since it would not be cost-effective to replicate a full-sized production system multiple times in support of development and testing, we designed OMIDAPS to function identically at whatever size we choose. Previous projects taught us that we need a system that will evolve as the mission unfolds. Since the growth direction is difficult to predict ahead of time, we devised a system that will grow in increments. The direction of each increment, whether it means adding storage, processing power, network hardware, or whatever, will be determined as the project develops. We start with a system that is small and out of balance, then add or reassign hardware without losing any of the initial investment.

Ellipses in Fig. 4 show some of the places where the system can expand. By adding or reassigning hardware, each OMIDAPS function can grown or shrink in performance. The system can be hosted across as many machines, and across as many disks as are suited for the processing job at hand.

Our design makes no assumptions about the number of servers in the system. As far as the software is concerned, there can be any number of any computer or disk. Database tables are populated so as to describe the hardware topology to the software. The tables provide pointers that show each software component where to find what across the network.

To demonstrate scalability, we have implemented OMIDAPS instances of varied sizes. Our smallest OMIDAPS runs on a single 1.2 GHz laptop. The database tables map all functions to the single computer.

The Test instance was brought up very early in the project, about two years ago. It was used for teaching our testers how to run OMIDAPS, and for exercising initial versions of OMI science code.

Our Production instance is expected to grow the most. As we progress through project phases, more machines will be added. Currently, we are eagerly awaiting the launch of Aura and first light data from OMI. The largest of our assembled systems, Production has been sized to process data during the first few months of the mission using science software that we have today. We expect this system to keep up with real time data from OMI at a nominal 1X processing rate. The system must actually support processing rates of at least 2X to accommodate periodic down time with the ability to catch up to real time. As we gain more science software to process,

we expect to grow the Production instance using hardware which will be more capable than what we could have bought initially for the same money.

Given real data, our science team will modify their algorithms during the first months after Aura's launch to better process whatever was unexpected from the instrument in orbit. We have been using simulated OMI data in lieu of real data; it proved to be a very important ingredient in the development of the science software. But simulated data does not have enough information and artifacts in it to permit complete development of the science algorithms. So starting a few months after the OMI launch, we expect a continuous stream of improved science code. After characterization and integration of the revised science code into OMIDAPS, and after evaluation of the integrated processing by the science team, we will rewind back to the start of the data collection and reprocess all of the mission's data with the new software. Algorithm improvement, and reprocessing, will continue throughout the mission.

To be able to quickly reprocess data, OMIDAPS has to have processing capability several times the nominal 1x rate. Experience from the SeaWiFS project and our own experience reprocessing MODIS, TOMS, and SBUV data shows that there is no upper limit on the desired x-rate for reprocessing. Faster is better.

Although faster and bigger is better in most cases, there is also a need for small OMIDAPS systems. It happens often that a scientist wants to reprocess some data subset in smaller volumes. Still too big for the scientist's development system, or too mundane to manually run thousands of times, some of our team members occasionally need to run jobs on an automated system rather than on a large or fast system. Sometime small and automated is also the right answer.

The Database Node is the only monolithic component of OMIDAPS that cannot yet be distributed across multiple hardware. We continue to closely monitor its performance as the system is scaled larger, but so far the database has not been a bottleneck for us. Experience reprocessing TOMS and SBUV data gave us confidence that our database is adequate. Using today's OMIDAPS, we reprocessed the complete TOMS and SBUV data sets several times. In the latest reprocessing of the TOMS data, OMIDAPS processed 108,000 full orbits (twenty-five years) of data through 4 main processing steps per orbit. The final result consisted of 800,000 L2 and L3 files which totaled about 400 GiB of product data. To produce it, there were about 350,000 discrete executions of the science software that were scheduled and managed by OMIDAPS. All this was done using a relatively small database server. The entire TOMS reprocessing took four days. That means the system handled about 88,000 science software executions per day using hardware that is already several years old.

OMI reprocessing will take longer than four days when all the mission's data becomes available because the OMI data rate is about 1,000 times that of TOMS. However, the number of algorithm executions that the database must support is roughly similar. With 15 Aura orbits per day, perhaps 6 years of raw OMI data, and about 60 algorithms, we expect a full reprocessing at the end of the mission to call for about two million algorithm executions. Using the 88,000 algorithm initiations per day from the current system that was used for the TOMS reprocessing, the database could handle two million initiations in about 23 days or less using today's hardware. Of course, with the higher OMI data rate, the overall reprocessing will take substantially longer than this. But the TOMS reprocessing proved that our system will be capable of scheduling and managing the needed number of jobs even for OMI.

Flexibility and growth of a system leads to an additional task: balancing all the resources. As hardware is added, we want all of the resources to be loaded at about the same level. No sense, for example, having the internal network extremely busy while disks and CPUs go mostly idle. Finding the right balance is discussed next.

B. Tuning

Balancing the cluster was discussed initially in a paper about MODAPS [2]. OMIDAPS presents similar challenges. The problem stems not so much from the architectural approach to building the system as it does from the nature of the work that we support. The system performs best when the various subsystems are configured optimally for the science software. Different remote sensing algorithms need different amounts of computer resources. OMIDAPS has to be configured accordingly.

Since our science software will change throughout the mission, balancing resources in the cluster will be an ongoing task. We are taking a multifaceted approach.

We start by taking a rough guess based on science software that we have in hand. Prior to 2003, simulated OMI data and the science algorithms were not ready for production testing. To configure, test, and tune the initial OMIDAPS prototype, we used 25 years of TOMS and SBUV data. The early OMIDAPS prototype not only provided for reprocessing and analysis of these historical data sets using improved algorithms, but it also gave us a platform that supported adaptation of science algorithms for OMI. It permitted the initial debugging of our techniques and processes. And it validated our new hardware architecture and approach.

Second, we trend the characteristics of the science software. The developers have provided early versions of their code. For over two years, we have involved the science team coders in integration of their software into early versions of OMIDAPS. The early and ongoing involvement has resulted in many versions of science code and OMIDAPS that have been integrated many times, each time with increasing level of complexity. The team has thus followed the spiral development model where we do the same develop-integrate-test-feedback cycle over and over while increasing maturity of the system each time.

As the cycle repeats, it has enabled us to streamline our processes. The software hand-off from developers, to integrators, to testers, to production has been smoother during each cycle. Although it took months for the first processing algorithm to

reach production, our goal is to shorten the cycle to just a few hours during post-launch algorithm improvements. Getting a new version of software quickly from developer into an instance of OMIDAPS is just as crucial as being able to reprocess large amounts of data quickly. Streamlining the integration permits rapid deployment of new code, and that permits faster system characterization and system re-tuning.

As new bottlenecks are identified, and as we grow the system, we will re-allocate existing components or purchase new hardware. The plan is to buy no more than what is needed for the evolving versions of the science code. So, for example, if we find that a new version of an algorithm needs much more temporary storage while it is running, we will supplement our processing nodes with more disk. If we find that some new version needs more CPU or RAM, we will add larger processing servers and configure tables so that the algorithm is allocated only to the larger servers for processing. If we find the revised algorithms producing much larger output files, we will add more disk to the Archiver, or increment the number of archiver hosts. The system can be expanded in many directions. The repetitive characterization, streamlined processes, and just-in-time hardware acquisitions will permit us to find the "sweet spot" of performance and balance the cluster.

C. Advantages

There are many advantages that stem from our generalized design and implementation. Here are just a few.

One major advantage stems from the timing of hardware purchases. Because we postpone buying components until we need them, we are able to acquire more capacity and performance for the same money compared to buying everything at the project's onset.

Another advantage comes from our generalized design and implementation. OMIDAPS does not depend on any unique feature of any vendor. It will run on a wide variety of hardware, whether Intel-based, Sun, HP, DEC, or whatever. As long as a computer can host Unix, OMIDAPS will run on it. OMIDAPS is implemented mostly in Perl using open source system software that is available for machines from a multitude of manufacturers. We took care to make the software portable. Since we will be acquiring hardware over the course of the project, the final system will likely end up very heterogenous. Not being tied to any particular hardware vendor permits us to shop for the best deal at the time. Since we will be purchasing increments over the course of many years, we will benefit from not being forced to choose the same vendor each time.

Another advantage to our approach results from the distributed, loosely coupled hardware. OMIDAPS is more resilient than the multi-CPU-in-one-box MODAPS. The whole OMIDAPS does not go down when a single component (CPU, memory module, power supply) breaks. MODAPS, on the other hand, grinds to a halt with most component failures. Multi-hour processing, if not completely finished, goes to waste. MODAPS can be often rebooted and run without the broken part, but when a replacement arrives, everything must be shut down again to install it. In contrast, the distributed OMIDAPS can have a broken host indicated with an entry in the database, the Scheduler stops initiating new jobs on it, and the rest of the system continues without a pause.

VII. Conclusion

An evolutionary development approach enabled us to build two very capable processing systems for remotely sensed data. MODAPS, that evolved out of a SeaWiFS system, went through several generations of architecture. Building on those experiences, we developed the latest incarnation of the system for OMI. While building OMIDAPS, we used it to perform a full reprocessing of twenty-five years of TOMS and SBUV data. OMIDAPS now stands poised to process and reprocess OMI data. For OMI, the production system will grow as the needs of the project grow.

We have developed a generalized architecture that can be used for building processing systems for diverse uses—not just for remote sensing. The design is capable of handling very large numbers of diverse inputs. It permits complex processing rules that provide for simultaneous and sequential combinations of processing. The processing system topology is easily set, scaled, and tuned by adding hardware and by changing entries in software lookup tables. Our system is capable and available to process data not only for OMI, but also for other coming missions.

Acknowledgment

The authors thank the many individuals comprising MODIS and OMI teams for making development of MODAPS and OMIDAPS successful. Some of the information presented here was taken from a number of documents and web sites throughout the two projects. OMIDAPS development is carried out under NASA contract number NAS5-00220.

References

[1] E. Masuoka, C. Tilmes, G. Ye and N. Devine, "Producing Global Science Products for the Moderate Resolution Imaging Spectroradiometer (MODIS) in the EOSDIS and MODAPS," Proceedings, IEEE Geoscience and Remote Sensing Society, 2000.

[2] C. Tilmes, E. Masuoka and P. McKerley, "Concepts for Scaling the Processing Capability of the MODIS Data Processing System (MODAPS)," Proceedings, IEEE Geoscience and Remote Sensing Society, 2000.