FAWKES Information Management for Space Situational Awareness

Scott Spetka
ITT Corp. and SUNYIT
George Ramseyer
Air Force Research Laboratory AFRL/RITA
Scot Tucker
ITT Corp

ABSTRACT

Current space situational awareness assets can be fully utilized by managing their inputs and outputs in real time. Ideally, sensors are tasked to perform specific functions to maximize their effectiveness. Many sensors are capable of collecting more data than is needed for a particular purpose, leading to the potential to enhance a sensor's utilization by allowing it to be re-tasked in real time when it is determined that sufficient data has been acquired to meet the first task's requirements. In addition, understanding a situation involving fast-traveling objects in space may require inputs from more than one sensor, leading to a need for information sharing in real time. Observations that are not processed in real time may be archived to support forensic analysis for accidents and for long-term studies. Space Situational Awareness (SSA) requires an extremely robust distributed software platform to appropriately manage the collection and distribution for both real-time decision-making as well as for analysis.

FAWKES is being developed as a Joint Space Operations Center (JSPOC) Mission System (JMS) compliant implementation of the AFRL Phoenix information management architecture. It implements a pub/sub/archive/query (PSAQ) approach to communications designed for high performance applications. FAWKES provides an easy to use, reliable interface for structuring parallel processing, and is particularly well suited to the requirements of SSA. In addition to supporting point-to-point communications, it offers an elegant and robust implementation of collective communications, to scatter, gather and reduce values. A query capability is also supported that enhances reliability. Archived messages can be queried to re-create a computation or to selectively retrieve previous publications. PSAQ processes express their role in a computation by subscribing to their inputs and by publishing their results. Sensors on the edge can subscribe to inputs by appropriately authorized users, allowing dynamic tasking capabilities.

Previously, the publication of sensor data collected by mobile systems was demonstrated. Thumbnails of infrared imagery that were imaged in real time by an aircraft [1] were published over a grid. This airborne system subscribed to requests for and then published the requested detailed images. In another experiment a system employing video subscriptions [2] drove the analysis of live video streams, resulting in a published stream of processed video output. We are currently implementing an SSA system that uses FAWKES to deliver imagery from telescopes through a pipeline of processing steps that are performed on high performance computers. PSAQ facilitates the decomposition of a problem into components that can be distributed across processing assets from the smallest sensors in space to the largest high performance computing (HPC) centers, as well as the integration and distribution of the results, all in real time. FAWKES supports the real-time latency requirements demanded by all of these applications. It also enhances reliability by easily supporting redundant computation. This study shows how FAWKES/PSAQ is utilized in SSA applications, and presents performance results for latency and throughput that meet these needs.

1. INTRODUCTION

Electro-optical sensors can operate at 250 Hz, and each image may be as large as 256 KB. Processing images at this frequency and of this size, for example to correct for blurring, is challenging. SSA processing demands were recently studied by Linderman et al[3]. Supercomputers are needed to support the parallel processing required to meet current and future requirements. A system that can distribute imagery to scientists and analysts must be carefully designed for optimal use of network and processing resources. New techniques are needed to meet performance requirements for streaming data from sensors and to make data streams accessible through networks. Point-to-point solutions do not scale well and present difficulties for developers. Since Internet Protocol (IP) multicast is User Datagram Protocol (UDP)-based, it is inherently unreliable.

The FAWKES high performance information management middleware can be used to support processing and distribution of imagery streams produced by sensors, to meet SSA demands. FAWKES implements a *brokering* function which routes information from publishers to subscribers, avoiding the need to establish direct connections between them. Metadata, which can be expressed in XML, is used to match publishers with subscribers. FAWKES is an implementation of the Phoenix information management architecture [4] under development at the Air Force Research Laboratory (AFRL). Earlier instances of this evolving high performance implementation of the information management middleware supported fault-tolerance [5] and interprocessor communications in parallel computations [6].

Section 2 shows how the current implementation of FAWKES supports communications and services for the In-line Speckle Image Recovery Engine (INSPIRE). INSPIRE is a system that provides a corrected stream of imagery from telescopes at the Maui Space Surveillance System (MSSS) [7] to analysts interested in viewing the images. Section 3 presents experimental results for this architecture. A further description of how the FAWKES/INSPIRE architecture can take advantage of resources that are geographically distributed is presented in the Future Large Scale FAWKES/INSPIRE Systems in Section 4. The advantages of this architecture are presented in the Conclusions, Section 5.

2. FAWKES/INSPIRE ARCHITECTURE

FAWKES supports the communications requirements of the INSPIRE architecture. Fig. 1 shows the overall flow of information through this system. The Real-time Object and Atmosphere Reconstruction System (ROARS) is a graphical front-end to INSPIRE, and is utilized to collect data from users. ROARS is a flexible system in its own right. ROARS enables the execution of codes by providing the capability to select streams of data to be processed, codes to be executed, and the required parameters used to drive the code. The code that is for these experiments is the Physically Constrained Iterative Deconvolution (PCID) algorithm [8], which is capable of running in parallel and supports many input parameters. ROARS itself also provides multiple inputs to support a wide range of functionality for end users.

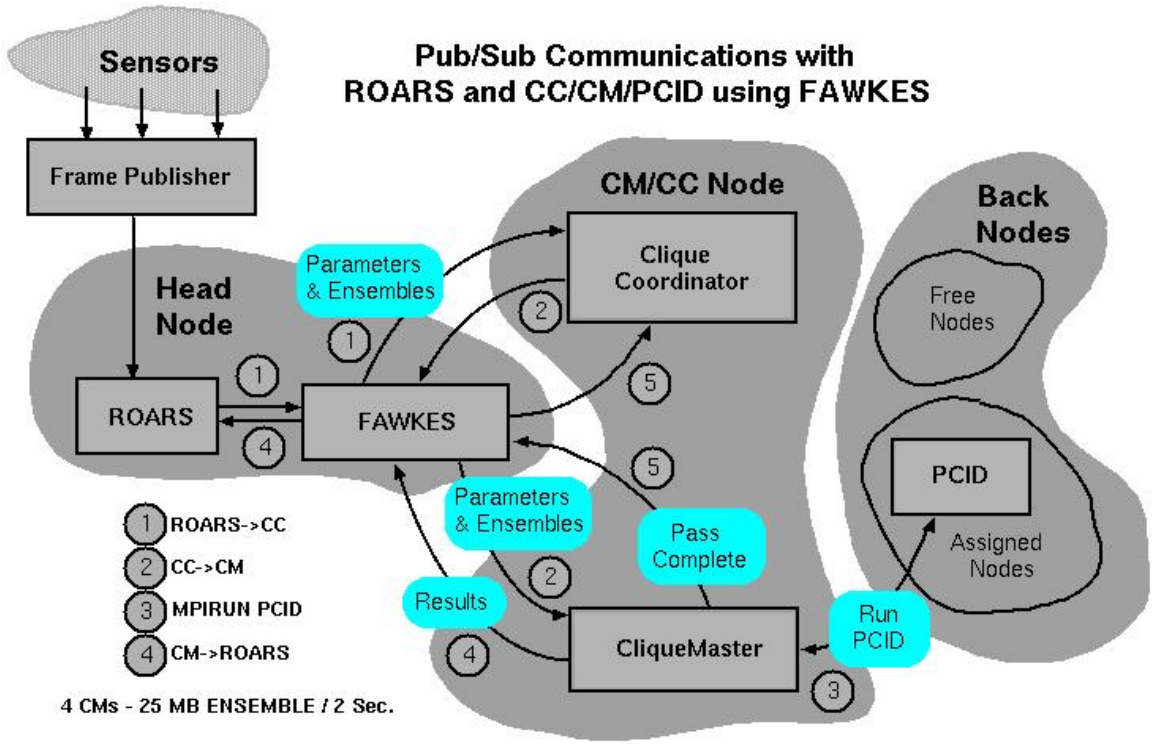

Fig. 1. FAWKES/INSPIRE Information Flow

FAWKES implements a pub/sub style of communication for information management that is characterized by decoupled senders and receivers [9]. Subscribers can specify publications to be received by content, based on the meta-data associated with the publications. Publishers submit the publications without regard to exactly which subscribers will receive them and are not even aware if a subscriber is currently listening for the new publications. The lack of dependence on specific addresses to establish communications channels has advantages for fault tolerance and location independence, since redundant subscribers may listen to a channel and subscribers may move without affecting ongoing communications.

The components of the INSPIRE system are presented in Fig. 1, along with the paths by which information flows from sensor inputs to the finished product display. On the front-end of the system sensor inputs can be collected directly by ROARS or published by a *Frame Publisher*, making them available to multiple users who are each running ROARS, allowing each user to process their streams with different codes or different parameters.

Processing requests sent by ROARS to the CliqueCoordinator (CC) include ensembles of raw frames that are to be processed to produce a single corrected (deblurred) frame and the parameters to be used by PCID for the necessary processing. As shown in Fig. 1, FAWKES and ROARS normally run on an HPC's head node. Requests for processing are submitted by Path 1 with ROARS interacting with the CC through FAWKES. ROARS assembles the sensor data into *information objects* (IOs) called ensembles, with associated XML metadata and input parameters, and submits the IOs to FAWKES. FAWKES *brokers* the IOs, forwarding them in this case to the CC. The process of brokering involves examining the XML metadata in the IO to determine which subscribers should receive it.

The CC manages a set of interactive processing nodes (back nodes) allocated from the HPC. Processing cores are allocated from the node pool for use in processing each requested PCID execution, similar to how time slices are allocated to processes by a time-sharing system. In the current implementation, nodes that are allocated by the CC are grouped into cliques of nodes, under the control of the ClickMasters (CMs), and are allocated statically to the CMs. CMs perform a launching function, analogous to the operating system function that initiates execution of binaries on a system.

The CC publishes the ensemble and parameters for processing to an available CM through FAWKES which brokers the IO's, as shown in Path 2. Through Path 3, the CM transmits the data to the assigned back nodes, where the data is processed, after which the results are transmitted back to the CM. In this case the CM uses the *mpirun* command to initiate the execution of PCID on the back nodes, since PCID uses Message Passage Interface (MPI) internally for communications among parallel processing cores. Through Path 4 the CM publishes the results to FAWKES, which brokers the IO's and then sends them to ROARS. The CM then publishes a *done* message to FAWKES by Path 5, which is received by the CC. This notifies the CC that the respective CM and its back nodes are now available for processing new incoming requests.

This architecture has been experimentally evaluated, and the results are presented and discussed in the next section.

3. RESULTS AND DISCUSSION

This study focuses on the information flow within this architecture. Experimental data also shows the contribution to processing time for PCID, and its impact on the total end-to-end latency. Two experiments demonstrated the behavior of the system under increased load. After discussing the experimental environment and presenting the PCID results, the paths presented in Fig. 1 are discussed and quantified, resulting in a complete evaluation of this architecture's FAWKES communications system.

For these experiments 20 ensembles of data were input into the system presented in Fig. 1. One node was used for executing the CC and four CMs. Two additional nodes with 8 cores each were used to implement four processor cliques, with each clique having 4 cores. Each of the 4 cliques was assigned to a respective CM. The experiments were performed at the Maui High Performance Computing Center [10] on MANA, a dual-quad Xeon Nehalem HPC system with 24 GB RAM on each node. PCID has been optimized to run in parallel, with runtimes reduced from hours to seconds. The experimental results presented in tables I and II are of the actual PCID version 9.1 timing. They do not include the overhead for the *mpirun* command that were used to launch PCID for each ensemble.

A legacy PCID benchmark is to process 400 frames of data. In the first experiment ensembles were 400 frames, with an ensemble input rates of 1 ensemble/2 seconds. The PCID input ensemble size for the experiment was 25 MB (400 frames, each 128x128 pixels). Every two seconds one of four CMs launched a PCID run on its respective backnodes. The results of this experiment are presented in Table I. The input parameters for PCID were adjusted to produce execution times of less than four seconds, to emulate the execution times that we expect from PCID version 9.1 when experimenting with a clique size of 80 to 104 cores.

The times for PCID processing the ensembles ranged from 3.521 to 3.771 seconds. The CC always found an available CM to assign for processing each incoming ensemble. This was a successful run that exceeded the experimental requirements.

Table I. 20 Ensembles /2 seconds

CM_Run Number	Last PCID Iteration Wall Clock Time	PCID Runtime (sec)
run0_0	**10:58:13.164**	**3.727**
run1_0	**10:58:16.909**	**3.656**
run2_0	**10:58:18.293**	**3.672**
run0_1	**10:58:21.646**	**3.716**
run3_0	**10:58:22.936**	**3.614**
run1_1	**10:58:25.669**	**3.717**
run2_1	**10:58:27.227**	**3.566**
run0_2	**10:58:30.175**	**3.597**
run3_1	**10:58:31.957**	**3.709**
run1_2	**10:58:34.890**	**3.697**
run2_2	**10:58:36.662**	**3.689**
run0_3	**10:58:39.686**	**3.771**
run3_2	**10:58:40.508**	**3.521**
run1_3	**10:58:43.589**	**3.561**
run2_3	**10:58:45.062**	**3.537**
run0_4	**10:58:48.187**	**3.576**
run2_4	**10:58:51.725**	**3.633**
run1_4	**10:58:52.211**	**3.564**
run3_3	**10:58:55.530**	**3.612**
run0_5	**10:58:56.806**	**3.578**

A more recent PCID benchmark is to processes a stream of images collected at 250 Hz, which requires a four second or less latency when using 4 CMs. In this experiment each ensemble was 250 frames, with a new ensemble every second. Four CMs, configured the same as those used in the first experiment, were used. In experiment 2, the input ensemble size was set to 15 MB (250 frames, each 128x128 pixels). Additional adjustments were made to the PCID input parameters for this experiment, to reduce the execution time of PCID further, while still running on four cores. The results are presented in Table II.

The times for processing the ensembles ranged from 2.432 to 8.706 seconds, with two of the runtimes exceeding the required latency. Because there was not a CM available, 6 of the ensembles were not launched. The smaller ensemble size and reduced PCID latency did not compensate for the increased arrival rate of ensembles from the sensor to the system. This was a partially successful run in which 14 of the 20 ensembles were processed.

Table II. 20 Ensembles /1 second

M/Run Number	Last PCID Iteration Wall Clock Time	PCID Runtime (sec)
run0_0	**11:20:51.915**	**2.514**
run2_0	**11:20:54.080**	**2.498**
run1_0	**11:20:55.452**	**2.508**
run0_1	**11:20:57.946**	**2.488**
run1_1	**11:21:00.926**	**2.468**
run3_0	**11:21:02.151**	**7.053**
run2_1	**11:21:03.727**	**8.706**
run0_2	**11:21:04.215**	**2.544**
run3_1	**11:21:07.683**	**2.432**
run1_2	**11:21:07.691**	**2.458**
run2_2	**11:21:10.138**	**2.435**
run0_3	**11:21:11.174**	**2.465**
run3_2	**11:21:14.011**	**2.378**
run1_3	**11:21:14.675**	**2.481**

These two experiments illustrate the capabilities and limitations of this current implementation. As the load on the system is increased, variations in end-to-end processing time can occur. Two of the PCID runs in the second experiment, took three times as long as the other successful runs, which appears to have destabilized the system. The results show increased instability as the rate of input approaches the capacity of the system. In the INSPIRE architecture, variations in the cost for communications or for PCID execution can result in longer end-to-end latency and the need to configure additional CMs. Fortunately, PCID latency is normally very consistent, varying mostly due to differences in input data.

The effect of variations in PCID latency in the 20 ensembles/1 second experiment was amplified in the overall time required to complete the *mpirun* command, to initiate PCID and wait for completion of the job. Further experiments are needed to determine which resources caused the variance in runtime under increased load and if it was due to contention by multiple copies of PCID. For these executions, each PCID had its own ensemble data as well as input file. However, it is unusual to run multiple copies of PCID on a single node, as we did in these experiments, since a single parallel execution of PCID would normally use more than one node.

The timing of the first four paths in the system shown in Fig. 1 is presented in Table III for the 20 Ensemble/ 2 second experiment. Additionally, the overall end-to-end timings are presented, measured from when a new ensemble was introduced into the system (Start of Path 1), until it was determined that the processing was complete (end of Path 5). The *mpirun* PCID times of Path 3 are longer than the actual PCID run times, as they include not only the PCID run time, but also the time to launch the PCID job on the back nodes using *mpirun*. Even when

including these launch times, and the times of the rest of the system, the actual complete run was a success, and all 20 ensembles were successfully processed. The system is however close to its capacity, and for further increases in throughput, additional CMs would need to be employed.

Table III. Overall system timing, 20 Ensembles /2 sec

Path	Component Flow	Time (sec)
1	ROARS to CC	0.298 - 0.312
2	CC to CM	0.509 - 0.520
3	MPIRUN PCID	3.96 - 7.67
4	CM to ROARS	0.008 - 0.020
	End-to-End	4.93 - 8.57

Additionally, the experiment shows that INSPIRE is close to meeting the overall objective of processing sensor data at 250 Hz. Meeting performance goals may require implementing CMs that pre-load codes to avoid contention that may be occurring even for the small number of nodes used in this study. Codes could also be prepositioned into local filesystems, like /tmp, to achieve a similar effect. The flexibility of FAWKES information management will also help in adapting the current design to meet these INSPIRE goals for end-to-end latency.

4. FUTURE LARGE SCALE FAWKES/INSPIRE SYSTEMS

FAWKES information management flexibility provides the basis for implementing geographically distributed systems for processing streaming sensor inputs. Fig. 2 shows an extension of INSPIRE using FAWKES. In the diagram FAWKES proxy services are used to cross HPC boundaries. Proxy services will support interactive access to resources at three processing sites, shown in the diagram. The diagram shows a user at Rome starting ROARS on the 500 TFlop heterogeneous Intel/Cell-BE/GPGPU cluster. A CC is automatically started when the nodes are allocated locally. The Rome CC publishes its status so that other CCs that are interested in the codes supported by the Rome CC may access the Rome nodes. Since all communication is through FAWKES, CCs and CMs can be accessed transparently from any location.

As the diagram shows the Maui High Performance Center (MHPCC) will add additional nodes for management by the CC. The CC subscribes to notices that additional nodes are available and adds them to available nodes. Static CMs are started, with nodes assigned to each, in the current implementation. Dynamic management of node allocation allows for processing varying size ensembles and varying parameter settings can also be implemented. New allocations are also published, so that other CCs can publish work for them. ROARS subscribes to CC publications according to user preferences as to processing sites. Inter-CC operations will require access controls in most cases, to assure the autonomous control over HPC system resources. For example, ROARS could request that a local CC not publish its resources for access by inter-HPC access.

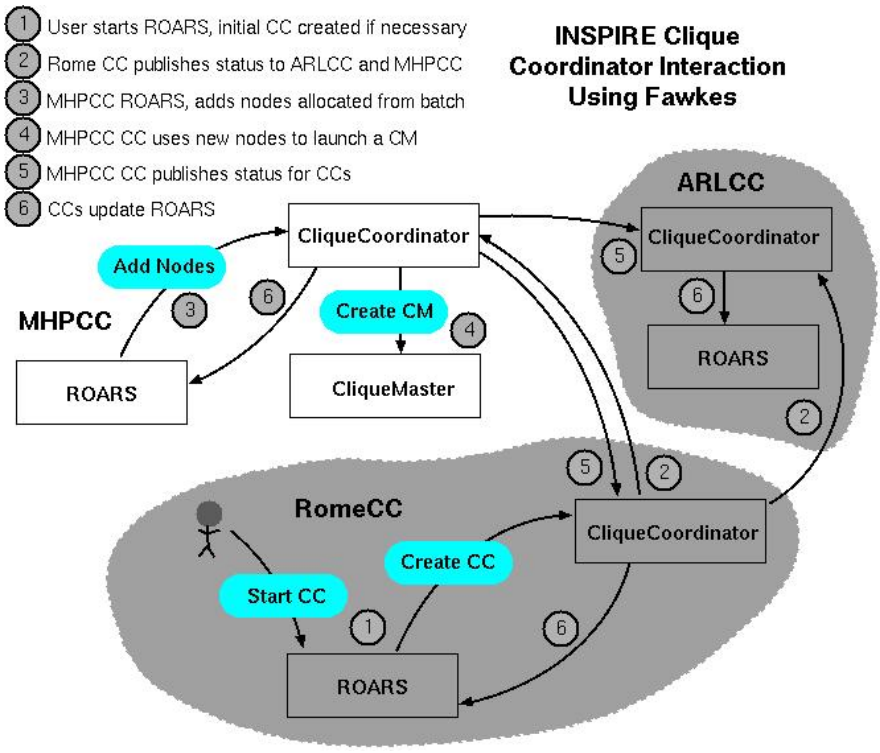

Fig. 2. Inter-HPC Communication Architecture with FAWKES

5. CONCLUSIONS

FAWKES has demonstrated the capacity to support the requirements of INSPIRE, a space situational awareness application that manages streams of incoming sensor data by distributing the data to HPC processing resources and publishing results for end users. The current implementation is very close to meeting performance goals for real-time processing. Approaches to improving performance to meet those goals were presented. FAWKES simplifies communications for inter-HPC access, allowing users to take advantage of remote processing resources. It also makes sensor inputs available to authorized users anywhere in the world, through FAWKES. The FAWKES middleware simplifies the application development interface, relieving developers of details associated with machine addresses, memory allocation, data security, etc. FAWKES code is portable to any system that implements the FAWKES services.

The advantages of implementing FAWKES between clients and services are enhanced configurability, reliability, productivity, fault tolerance and scalability. Integrating code is simplified, and additional codes are more easily integrated. The management of connections to multiple locations is simplified, and these connections may come and go on either the client or server side.

Other middleware advantages afforded by FAWKES include independence from the underlying communications layer. This allows FAWKES to be portable across network fabrics. For example, FAWKES may access native InfiniBand (IB) while avoiding the overhead of TCP over IB. Also, FAWKES serialization alleviates architecture dependence on data formats and byte orders. FAWKES *proxy services* use SSH tunnels to provide conduits among HPCs that allow cliques to operate on multiple HPCs.

By integrating ROARS with FAWKES, mission-critical codes can be run on two or more CMs, assuring results from at least one. ROARS could ignore duplicate results, or it could simply display all of the results without change. ROARS publishes requests to the CC without needing any location service or knowledge of CC location, making configuration easier, since *config* files or environment variables are not needed. ROARS is able to subscribe to results without knowing who is publishing them. A ROARS subscriber that is publishing ensembles and subscribing to results would not be aware if a CM was shut down and replaced by a CM running on a different machine.

6. ACKNOWLEDGEMENT

Special acknowledgements are given to the INSPIRE team: Kathy Borelli, Adam Mallo, Jason Addison, Brad Farnsworth, Ron Viloria, Lisa Thompson, Bruce Duncan and Chris Sabol for their contributions.

7. REFERENCES

1 R.W. Linderman, J. Corner and S. Tucker**, "**Swathbuckler: Real-Time Wide Swath Synthetic Aperture Radar Image Formation using Embedded HPC", Proc. HPCMP Users Group Conf., pp. 244-241, Jun. 2006.

2 L.K. Yan, G.O. Ramseyer, R.W. Linderman and E. Balster, "Video Teleconferencing on the Distributed Interactive HPC Testbed (DIHT)", 2005 DREN Networking & Security Conference, Charleston, SC, Oct. 17-21, 2005.

3 R.W. Linderman, S.E. Spetka, S. Emeny and D. Fitzgerald, "Enhancing Image Processing Performance for PCID in a Heterogeneous Network of Multi-code Processors", Advanced Maui Optical and Space Surveillance Technologies Conference, Wailea Beach Marriott Resort, Maui, Hawaii, Sep. 1-4, 2009.

4 Rob Grant; Vaughn Combs; Jim Hanna; Brian Lipa; Jim Reilly, "Phoenix: SOA based information management services", Proc. SPIE, Vol. 7350, 73500P (2009); doi:10.1117/12.817911, Defense Transformation and Net-Centric Systems 2009, Orlando, FL, 14 April 2009.

5 S.E. Spetka, G.O. Ramseyer, and R.W. Linderman, "Fault Tolerant Integrated Information Management Support for Physically Constrained Iterative Deconvolution", Applied Imagery Pattern Recognition (AIPR) Workshop: Multiple Image Information Extraction, Cosmos Club, Washington DC, Oct. 15-17, 2008.

6 S. Tucker, S.E. Spetka, G.O. Ramseyer, S. Emeny, D. Fitzgerald and R.W. Linderman, "High Performance Information Management for HPC Parallel Computing," hpcmp-ugc, pp.409-412, DoD HPCMP Users Group Conference, 2008.

7 Maui Space Surveillance System (MSSS) - http://www.maui.afmc.af.mil

8 Charles L. Matson, Charles C. Beckner, Jr., Kathy Borelli, Tom Soo Hoo, Shiho You, Brandoch Calef, Maria Murphy, Ron Viloria, **"**PCID and ASPIRE 2.0 – The Next Generation AMOS Image Processing Environment", Advanced Maui Optical and Space Surveillance Technologies Conference, Wailea Beach Marriott Resort, Maui, Hawaii, 2007.

9 S.E. Spetka, S. Tucker, G.O. Ramseyer and R.W. Linderman, "Imagery Pattern Recognition and Pub/Sub Information Management," *Applied Imagery Pattern Recognition Workshop, 2007. AIPR 2007. 36th IEEE* , pp.37-41, 10-12 Oct. 2007.

10 Air Force Maui Optical and Supercomputing Site, http://www.maui.afmc.af.mil