Redefining Tactical Operations for MER using Cloud Computing

Joseph C. Joswig and, Khawaja S. Shams
Jet Propulsion Laboratory/California Institute of Technology/NASA
4800 Oak Grove Drive
Pasadena, California 91109
Phone (818) 298-4834
Fax (818) 393-5074
Joseph.C.Joswig@jpl.nasa.gov and Khawaja.S.Shams@jpl.nasa.gov

Abstract—The Mars Exploration Rover Mission (MER) includes the twin rovers, Spirit and Opportunity, which have been performing geological research and surface exploration since early 2004. The rovers' durability well beyond their original prime mission (90 sols or Martian days) has allowed them to be a valuable platform for scientific research for well over 2000 sols, but as a by-product it has produced new challenges in providing efficient and cost-effective tactical operational planning. [***]

An early stage process adaptation was the move to distributed operations as mission scientists returned to their places of work in the summer of 2004, but they would still came together via teleconference and connected software to plan rover activities a few times a week. This distributed model has worked well since, but it requires the purchase, operation, and maintenance of a dedicated infrastructure at the Jet Propulsion Laboratory. This server infrastructure is costly to operate and the periodic nature of its usage (typically heavy usage for 8 hours every 2 days) has made moving to a cloud based tactical infrastructure an extremely tempting proposition.

In this paper we will review both past and current implementations of the tactical planning application focusing on remote plan saving and discuss the unique challenges present with long-latency, distributed operations. We then detail the motivations behind our move to cloud based computing services and as well as our system design and implementation. We will discuss security and reliability concerns and how they were addressed.

TABLE OF CONTENTS

[*] 978-1-4244-7351-9/11/$26.00 ©2011 IEEE
[*] IEEEAC paper #1652, Version 3, Updated January 11, 2011
[*] This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

1. INTRODUCTION

Activity Planning for the Mars Exploration Rover Mission (hereafter MER) is a time consuming and intensely detail-oriented process in which teams of scientists and engineers collaborate to develop the best plan for upcoming planning cycle. Due to the communications delay of up to 20 minutes between Earth and Mars, operations are planned out for a complete sol or even multiple sols in an iterative process. The planning team balances scientific priorities from various theme groups such as Atmospheric Science and Geology with underlying rover safety constraints such not exceeding limits on power usage and on-board memory consumption. The end result is an activity plan that contains the list of activities and parameters that the robot will be instructed to perform such as acquiring imagery, driving to selected goals, usage of individual instruments, as well as engineering tasks.

Early MER mission planning was done using Jet Propulsion Laboratory's The Science Activity Planner, the 2004 co-winner of NASA Software of the Year. The Science Activity Planner (hereafter SAP) is used for both downlink data analysis and uplink activity planning [1]. Data analysis is focus on image and mosaic browsing and annotation with mission targets. Activity planners used SAP onsite at JPL to develop and refine rover plans as a centralized team [2]. As the mission's success continued, critical parties were eager to return to their respective home institutions and the development of a distributed solution was required. One trivially simple option that has been used on a limited basis is desktop sharing, which allows the user's to leverage existing 3rd party software to connect to JPL machines over VPN and use the centralized software application without any modification. The downside of such a system is potentially poor performance for users based long distances away from Pasadena, CA be it stationed across the country in New York or even internationally. The high latency these users experience is unavoidable, and particularly not user-friendly, so instead a client – server system, centered on Maestro for MER, was built to better support distributed operations.

Maestro for MER is the thick client desktop application built as the successor to SAP. Maestro for MER offers

improved downlink and uplink capabilities compared to SAP while having low hardware requirements such that users can run it on their laptops so long as they have an active network connection to JPL. Maestro for MER is built on the Eclipse Rich Client Platform that supports deployment on multiple platforms. Maestro's initial capabilities included activity plan editing, image and mosaic searching and viewing, as well as targeting which allows the users to pick out points of interest. Over it's lifetime Maestro's downlink capabilities have been updated to include support for new features such as overhead mapping provided by the HiRISE camera on the Mars Reconnaissance Orbiter to aid in creating more accurate rover maps then previously possible [3]. Additionally the Activity Planning / uplink capabilities have been improved with support for follow along planning, but one area that was lagging behind was Plan Saving and Retrieval, particularly for remote users. In this paper we will focus on the various implementations we've utilized for the saving, searching, and loading of activity plans over the course of the mission.

2. ACTIVITY PLANNING

Plan Model—Activity Plans are the result of the planning process. Plans composed of top-level meta-data such as modification time, author, and plan name as well as the content of the plan which is a list of Activity Groups [4]. Activity Groups are named containers that have zero or more Activities. Activities correspond to the individual actions that the robot will be instructed to perform. Some types of activities are image captures, robot drives, and instrument power activation. These activities then have dozens of individual parameters, which specify the exact behavior. In addition to their parameters, the resource usage of each activity is also shown. The exact details of the structure's hierarchy are not of particular importance, but the Object Oriented design context is critical to understanding the pros and cons detailed below in database storage example.

1) Activity Plan
   a) Meta Data
   b) Activity Group 1
      i) Activity 1
         (1) Parameter 1
         (2) Parameter 2
      ii) Activity 2
   c) Activity Group 2

Plans are created in a custom table editor that provides the user with an overview of the resources usage of individual components and the plan as a whole. The contents of the currently loaded activity dictionary are shown in a tree view and the user can drag and drop individual activity definitions from the dictionary into currently open plans.

File Based Persistence—SAP was initially developed to run locally on JPL's workstations in the confines of the Science Operators Working Group (SOWG) meeting room. Due to this configuration the most direct mechanism was file system based persistence. All machines were on the same network so they used NFS (Network File Systems) to save the plans to disk. Plans were serialized to Rover Markup Language (RML), an XML based interchange format, using Castor. This system had high performance due to locality to the storage medium. Plans were stored in a well-defined structure based on meta-data such as what sol they were used on, but there was no additional integrated search capability.

Hibernate Database Persistence—As previously mentioned the impetus for developing Maestro for MER was tied to the need to support distributed operations. A case study of potential solutions at the time led to the decision to store Activity Plans in a relational database and use Hibernate to perform the object-relational mapping. Using a relational database for persistence had additional benefits in that plans could be easily searched via SQL as well as precisely updated. Hibernate's typical mapping of objects creates a database column for each primitive (Integers, String, floating point numbers, and Boolean) field in a class. Fields that are complex objects and don't have a direct mapping to an existing SQL type are referenced using a one-to-one or many-to-one reference as appropriate using a lookup table. This schema design effectively means that each activity plan is stored, not in a single database tuple, but rather in hundreds of interconnected tuples since Activities, Activity Groups, Plans, and Parameters each are stored in separate tables. Saving a plan meant making serial connections to the database server and executing one SQL statement for each tuple. Activity Plan integrity was ensured using standard relational database concepts such as transactional reads and writes and foreign key constraints.

A major improvement in moving from SAP to Maestro for MER was the addition of support for Plan Searching. As Maestro was being designed the MER mission was continuously generating new plans which made it harder to locate an individual plan from the past if the user didn't know off-hand the specific details of when it was created and what stated it was last in, since those two criteria had determined the path of the file system

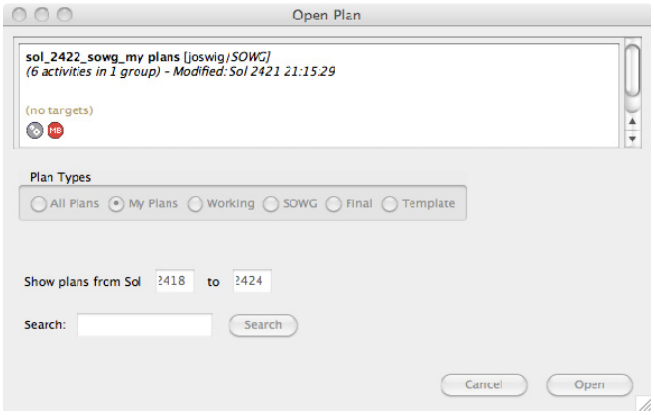


Figure 1: MER Plan Search Dialog

Implementing Plan Searching was a natural fit with the move to relational database storage since we could write custom queries to target specific fields in the plan model and allow users to look for plans containing references to specific targets and use of particular instruments.

One unique trait of MER planning is the iterative process of plan refinement and the accompanying roles and plan states. [4] Starting with a standard plan template the SOWG members go through various stages where they develop a

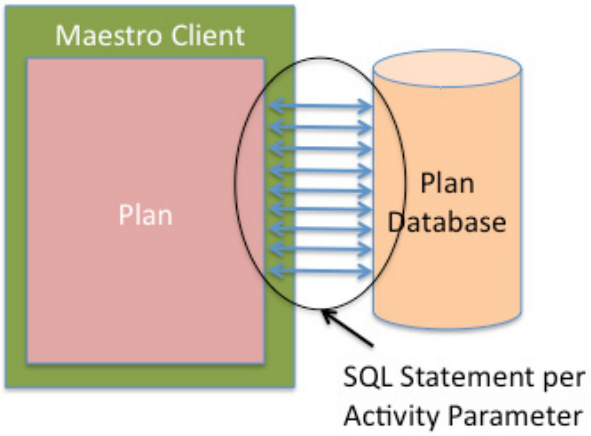


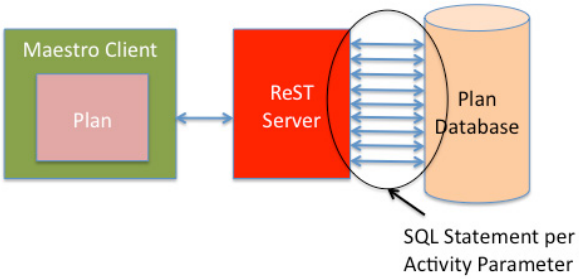
Figure 2: Direct Database Persistence

skeleton of the day's activity together, then separately and in parallel the various instrument leads will then fine tune Activity Groups relating to their instrument of concern. In this 'Refinement' stage the instrument leads will be restricted from structural modifications to the plan and can only modify the parameters of the activities in the groups that they own. When users have completed their individual refinement they then perform a partial plan save which updates only the database tuples for which they have write permission. This partial plan saving process fits naturally with the flattened database structure as we can precisely update the appropriate activity parameters and we don't need to concern ourselves with conflicts from multiple temporally close plan saves since each user will be updating a mutually exclusive component of the plan. There is still the requirement on a separate capability for the users to receive notification of external changes and this is accomplished using JMS.

The Hibernate based approach was an effective implementation and allowed for the newly added requirement of remote operations as well supporting the newly added functionality such as partial plan saves and plan search, but it had an unanticipated flaw. Once we moved to supporting remote users, we noticed the dramatic effect high latency connections had on Plan Saving and fetching. The serial nature of the tuple updates meant that if a user had a 25ms latency to JPL and their plan save touched 300 tuples the save would take almost 8 seconds to complete. If that was the worst case it likely would have been acceptable, but MER users were accustomed to creating template plans which contained several dozen activities and in excess of 1000 individual parameters and only exacerbated this performance bottleneck. The next step was to ameliorate this worst case.

Hibernate Database over ReST Persistence—The previously mentioned worst-case performance problem became quickly evident as Maestro received wider adoption from the distributed science community. One of the benefits of our original design was that we had a smart client-dumb server approach so there was little overhead in terms of server maintenance. Our existing institutional Database SA could perform backups and restores as well as monitor the server's uptime., but we needed a better option. Our next improvement was to turn to Representational State Transfer (hereafter ReST) and built a smart server. Since the high latency was the biggest bottleneck we designed a ReST server that sends and receives the plan as serialized binary blob over VPN. The server has now the single point of contact is the existing plan database and their location on the same network ensures low latency and significantly reduced save and load times.

Figure 3: Plan persistence using a ReSTlet intermediary



The Maestro client was updated to contact the ReST server via HTTP Put, Get, and Delete for creating, fetching and deleting plans. Functionally the capabilities remained the same, but with improved performance at the cost of additional maintenance overhead as a separate server

needed to be built and deployed with each update. The ReST based architecture improved the user experience significantly for remote users, but recent technological developments have led to even higher performing options.

3. CLOUD PERSISTENCE

As mission operations have continued the size of the planning database has continued to increase. Additionally the operations process has been streamlined so that multi-sol plans are now the norm. The combination of longer and therefore larger plans, plus data accumulation over time created more total data in the database, and evolving technology has led us to look at new options. The Maestro team has started optimizing the planning software responsible for the bottlenecks in the pipeline. Our solution involves novel use of cloud computing to optimize the process, which enables us to elastically expand and contract the resources available to our database on demand, while only paying for the compute capacity we utilize. Our solution decouples the meta-data about the plan from the actual plan data and stores them separately. The plans are stored as blobs in Simple Storage Service (S3), while the metadata is indexed in SimpleDB. With this design, plan searches resolve within tens of milliseconds, including network latency, while the largest plans can be retrieved within a matter of a few seconds.

The unique solution effectively leverages cloud computing to deliver performance and scalability. For indexing the plans, we utilize SimpleDB, a database service provided by Amazon Web Services (AWS). SimpleDB is a document oriented database that provides a Restful interface to our data, while offering scalable searching mechanism. JPL only pays for the storage of our data and for exactly the cycles used by our queries. Furthermore, SimpleDB automatically scales up the capacity for our database whenever it faces heavy queries or saves. This enables us to provide a streamlined operations even in the busiest of times. This database enables us to quickly search across thousands of plans without opening a single one to obtain all of their associated metadata. This is a great fit since plan searching and saving tend to be very temporally clustered and bursty operations centered around the planning cycle's timeline.

The plans are stored in S3. S3 is a scalable data store that is designed to handle heavy traffic. Furthermore, S3 replicates our files to ensure redundant copies are available in case of a storage failure. S3 also provides a Restful interface to all our files that enables us to write simple programs to download, upload, or delete files in the data store using standard HTTP methods PUT, POST, DELETE, GET and LIST. S3 provides very fast performance that enables us to download the largest plans within a matter of seconds.

Our solution provides a cost effective mechanism to obtain revolutionary optimizations in the plan saving, retrieving, and searching features for MER. Our application marks MER as the first NASA mission to effectively leverage a commercial cloud for a production mission operations application. The approach offers drastic performance improvements over the previous design, built on MySQL, and offers enough scalability to allow MER to operate for tens of thousands of sols without any degradation in performance

Ensuring Consistency— Partial Plan saving in Maestro is a vital part of the MER planning process. It allows scientists to concurrently commit partial saves of the plans, where their changes are localized to only a small portion of the full plan, this limited save is done during the Refinement stages and saves are typically scoped by instrument team. Since the plans are bigger than the size limit imposed by SimpleDB, we must store these plans in S3. However, decoupling the plans with their metadata makes it impossible for us to make conditional puts on the contents of the plan to ensure that a commit is fully merged and does not override changes made by a different scientist.

The Planning Software group has devised a simple, yet elegant solution to solve this problem. Our solution stores a canonical copy of the blob in S3 before making a transaction in SimpleDB. Until the transaction succeeds, the blob is unreferenced, and it does not exist from the perspective of the application. The blob is named by the hash of its contents, and the database row contains the hash of the referenced blob.

When committing a plan to the database, we start by obtaining the latest copy of the plan from S3 as referenced by SimpleDB. In this process, we obtain the hash of the latest copy as well as the latest plan itself. We merge the local plan with the most up-to-date canonical copy as known by the database. Subsequently, we generate a hash of the local merged copy, and we store the file in S3 and name it by its hash. We then make a conditional PUT in the database for the same plan name, with the condition that the latest hash that the database knows about is the copy of the plan that we merged with. If the latest hash has changed, in the case of someone else committing simultaneously, we obtain the latest copy and retry our commit.

Our solution is fault tolerant. Consider the case where the application crashes after committing a blob to S3. This crash will not corrupt the database as this blob is yet to be referenced by the database. On the other hand, consider the same scenario if we committed the transaction in the database first. If we crash after committing to the database and before inserting the blob into S3, our database is now corrupt and cannot reference this non-existent blob. Therefore, it is crucial to our algorithm to commit to S3 prior to making the transaction with SimpleDB.

Our solution is fully transactional. Consider the scenario where during the commit, a different user succeeds in updating the plan while a slower client is still in the process of updating. Since our call to update the row contains a conditional PUT, the PUT will fail. This failure will cause our client to update the latest copy of the plan and merge with it before attempting to save again. Furthermore, in cases of failed transactions, our approach offers automatic rollbacks.

Our solution is self-maintaining. Upon a successful transaction, we erase all existing copies of the blob that were referenced previously at any point. This approach minimizes the chances of orphaned blobs. Furthermore, successful transactions erase all previous copies, which clears the blob store of all unreferenced data. Lastly, we can run a daemon that cleans up all unreferenced blobs that are older than a day to safely erase all excess blobs.

Performance-Cost Comparison—As we've moved to a cloud services based implementation we've seen a tremendous improvement in performance, particularly with

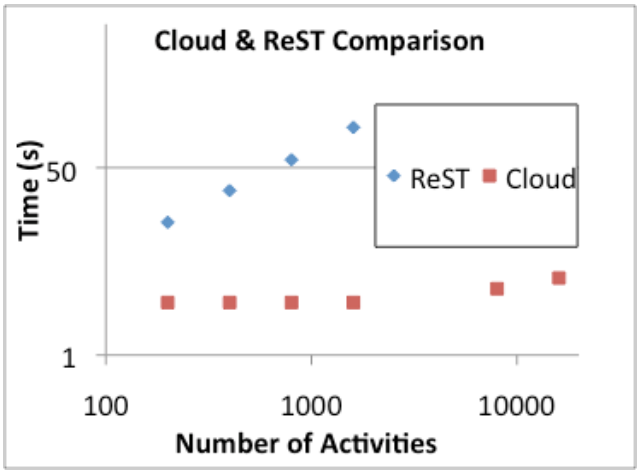


Figure 4: Comparison of Save Times – Both Axis are Logarithmic

larger plans. The following chart shows a comparison of save times for identical plans from the same remote machine on the same wireless network. Both axis are on a logarithmic scale. Both data series show a roughly linear growth in save times based on the number of activities and in turn activity parameters, and can be approximated with the following linear function $T_{Total} = A * C + T_C$, where A is the number of Activities, C is a constant factor and T_C is the non-network dependent processing time. C is empirically observed to be significantly smaller with the Cloud based implementation and the reason for our improvement.

In addition to much improved performance, cost was also a primary motivation to moving to a cloud services based architecture. Maestro's use of JPL's institutional services such as database administration and backup is summarized below.

Resource	Unit Cost	Total
2 ReST servers	$670/month/machine	$1340/month
Data storage of 2.5TB	$335/month/TB + backup: $500/month/TB:	$2000/month

Total Yearly Cost: ~ $40K.

Next we will provide a summary of our costs under the new architecture.

Resource	Unit Cost	Total
Storage – including backup	$165/TB/month	$412.50/month
Load balancer	$0.025 / hour	$18/month
1 Large machine running 24/7	$191/month	$191/month
10 Large machines 21 hrs/week	$34/month/machine	$340/month
Bandwidth – 300 GB outgoing	$0.15/GB	$45/month
Small instance at JPL	$70/month	$70/month

Total Yearly Cost: ~$13K

In this calculation we have included the cost of a constantly running large machine as well as 10 machines that will be provisioned during image processing. This conservative estimate of the cloud-based cost we are saving $27K annually on infrastructure costs while providing better performance to our end users.

4. Related and Future Work

Polyphony—Compared to plan persistence, image processing, particularly mosaic generation is an even more natural fit for moving to cloud based services. Mars missions such as MER, Phoenix, and soon MSL all produce large downlink products that are processed extensively to help the client better interact with them. This imagery is

downlinked from the spacecraft on a daily basis and efficiently creating usable client products helps provide the science planning team with new data at a rapid pace. One type of processing is the tiling of high-resolution images including the aforementioned mosaics. [5] We use tiled images so clients can lazily load only the resolutions and area they are currently viewing which minimizes the download time to first view an image while allowing for seamless zooming in when desired.

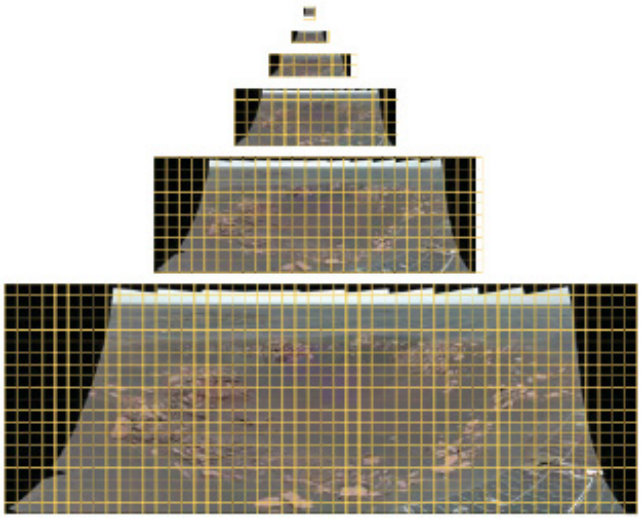


Figure 5: Polyphony Mosaic Tiles

Polyphony has been designed to manage the creation of mosaic and image tiles in a parallel process across multiple machines. Polyphony works by dividing up the work queues and then provisioning multiple machines to work on separate processing tasks in a divide and conquer approach. These machines utilize shared storage and don't have large memory requirements since they use the Kakadu JPEG2000 library to only load the portion of the images that is needed for manipulation.

Content Delivery Networks—One potential technology we will likely utilize going forward is cloud based content delivery networks. Since Maestro and its MSL counterpart MSLICE have globally distributed user bases it is important to provide speedy downloads to all corners of the Internet. CloudFront and similar technologies offer a simple interface for providing content to a broad user base. These content delivery networks provide low-latency edge nodes distributed in different locations so that instrument leads located abroad can quickly access the most popular plans and data products. We can heuristically determine which products and plans are of highest interest, at a first cut typically the most recent plans, and thus ensure that our users perceive minimal lag.

5. CONCLUSION

In this paper we summarize how the planning process has changed over time and why Cloud computing has proved to be an excellent fit for MER plan persistence. We have striven to balance implementation simplicity with our ever-evolving performance requirements. Our first Hibernate based implementation was designed with simplicity in mind and worked reliably save plans for the long latency remote user. Minimizing the number of round trips with the addition of the ReST was a simple improvement, but only an intermediate stage. The cloud based plan persistence implementation is the first use of Cloud computing on a flight mission and early user feedback has been very favorable. As we move forward in developing the planning and sequencing software subsystems for future missions, including Mars Science Laboratory, we will leverage these lessons learned and develop a similar architecture to provide similar cost-savings and performance benefits. Vendor lock-in is a viable concern with any application and while we haven't built a system that is immune to it, the cost to migrate to another cloud provider, such as Google and their BigTable datastore would be relatively minor. Our current active user base benefits greatly from the high speed and efficiency of these cloud services. Additionally, as a development team moving to a commodity server setup has greatly simplified the software update process.

REFERENCES

[1] Jeffrey S. Norris Mark W. Powell, Marsette A. Vona, Paul G. Backes, Justin V. Wick. "Mars Exploration Rover Operations with the Science Activity Planner". IEEE International Conference on Robotics and Automation. 2005.

[2] Jeffrey S. Norris, Mark W. Powell, Jason M. Fox, Kenneth J. Rabe, I-Hsiang Shu. "Science Operations Interfaces for Mars Surface Exploration". IEEE International Conference on Systems, Man and Cybernetics. 2005.

[3] Mark W. Powell, Thomas M. Crockett, Jeffrey S. Norris, and Khawaja S. Shams. "Geologic Mapping in Mars Rover Operations" American Institute of Aeronautics and Astronautics. 2009

[4] Michael McCurdy. "Planning Tools for Mars Surface Operations: Human-Computer Interaction Lessons Learned" IEEE Aerospace. 2009

[5] Khawaja S. Shams,, Dr. Mark W. Powell., Tom M. Crockett, Jeffrey S. Norris, Ryan Rossi, Tom Soderstrom "Polyphony: A Workflow Orchestration Framework for Cloud Computing". 2010.

BIOGRAPHY

Joseph C. Joswig *is a software engineer at JPL within the Planning Software Systems group. He has worked at JPL since July 2005. Recently his work has been focused on developing planning sequencing tools for Mars Science Laboratory as well as Emergency Response Software in support of the Department of Homeland Security. Past work has focused on operations for real-time and near real-time scenarios including JPL's ATHLETE (All Terrained Hex-Limbed Extra Terrestrial Explorer) robot and White Sands Missile Range's multi-asset tests. He received both his MS in Computer Science (2005) and his BS in Computer Science and Engineering (2003) from the University of California, Los Angeles.*

Khawaja S. Shams *is a member of the Operations Planning Software (OPS) Lab at the NASA Jet Propulsion Laboratory. At the OPS Lab, Khawaja develops software that contributes to the operations of a variety of robotic assets including ground, airborne, and waterborne robots, as well as robots on Mars. He leads a variety of software projects, and he serves as the Cognizant Engineer of server side components for the Activity Planning and Sequencing Subsystem (APSS) for the Mars Science Laboratory. Khawaja works closely with the Office of the CIO at JPL to co-lead the efforts to securely deliver the benefits of cloud computing to missions across NASA. He serves as an advisor on the CIO Technology Advisory Board (CTAB) at JPL. Khawaja obtained his bachelors in computer science from UC San Diego, and his Masters in Computer Science from Cornell. He is currently pursuing a PhD in robotics at USC under the advisement of Maja Matari.*