Leveraging the Cloud for Robust and Efficient Lunar Image Processing

George Chang, Shan Malhotra, Paul Wolgast
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr. Pasadena, CA 91101
{George.Chang, Shan.Malhotra, Paul.Wolgast}@jpl.nasa.gov

Abstract -- The Lunar Mapping and Modeling Project (LMMP) is tasked to aggregate lunar data, from the Apollo era to the latest instruments on the LRO spacecraft, into a central repository accessible by scientists and the general public. A critical function of this task is to provide users with the best solution for browsing the vast amounts of imagery available.

The image files LMMP manages range from a few gigabytes to hundreds of gigabytes in size with new data arriving every day. Despite this ever-increasing amount of data, LMMP must make the data readily available in a timely manner for users to view and analyze. This is accomplished by tiling large images into smaller images using Hadoop, a distributed computing software platform implementation of the MapReduce framework, running on a small cluster of machines locally. Additionally, the software is implemented to use Amazon's Elastic Compute Cloud (EC2) facility. We also developed a hybrid solution to serve images to users by leveraging cloud storage using Amazon's Simple Storage Service (S3) for public data while keeping private information on our own data servers.

By using Cloud Computing, we improve upon our local solution by reducing the need to manage our own hardware and computing infrastructure, thereby reducing costs. Further, by using a hybrid of local and cloud storage, we are able to provide data to our users more efficiently and securely. [1][2]

This paper examines the use of a distributed approach with Hadoop to tile images, an approach that provides significant improvements in image processing time, from hours to minutes.

This paper describes the constraints imposed on the solution and the resulting techniques developed for the hybrid solution of a customized Hadoop infrastructure over local and cloud resources in managing this ever-growing data set. It examines the performance trade-offs of using the more plentiful resources of the cloud, such as those provided by S3, against the bandwidth limitations such use encounters with remote resources. As part of this discussion this paper will outline some of the technologies employed, the reasons for their selection, the resulting performance metrics and the direction the project is headed based upon the demonstrated capabilities thus far.

[1] 978-1-4244-7351-9/11/$26.00 ©2011 IEEE.
[2] IEEEAC paper #1690, Version 1, Updated October 12, 2010

TABLE OF CONTENTS

1. INTRODUCTION 1
2. IMAGE TILING 3
3. PARALLEL COMPUTING APPROACH 4
4. CLOUD IMPLEMENTATION 5
5. CONCLUSIONS AND FUTURE WORK 7
ACKNOWLEDGEMENTS 7
REFERENCES 8
BIOGRAPHY 8

1. INTRODUCTION

The Lunar Mapping and Modeling Project (LMMP) is a NASA led initiative with the aim to create a single, consistent, uniform, intuitive and easy-to-use NASA portal. LMMP users consist of a number of communities including: current NASA lunar mission designers and science community, potential future lunar missions, international partners, commercial entities, education and public outreach. LMMP is tasked to aggregate lunar data from historical missions, international missions, currently flying missions and future mission data into a central state-of-the-art database and to make this data accessible to scientists and the general public. Particular data products include: lunar terrain and surface imagery, lunar gravity, lunar rock distributions, lunar crater distributions, lunar surface temperature, lunar mineralogy, and lunar lighting. The LMMP project is managed by MSFC and is being developed at JPL, GSFC, USGS, AMES and CRREL.

The LMMP system utilizes many open standards to allow access to the data. These standards conform to the platform-independent Open Geospatial Consortium (OGC) Standard Protocol including Web Map Service (WMS), Web Feature Service (WFS), Web Coverage Service (WCS), and Keyhole Markup Language (KML). Information is stored with metadata conforming to the Federal Geographic Data Committee (FGDC) standard. The LMMP Portal is expected to be operational in December 2010. It is implemented as a multi-tier system with a backend data management system, a

middle tier computation infrastructure and a lightweight, client-side Flash application front-end (see Figure 1). Users can use the interface to view maps, create complex overlays to analyze multiple maps, search for information, and download data directly to their computers.

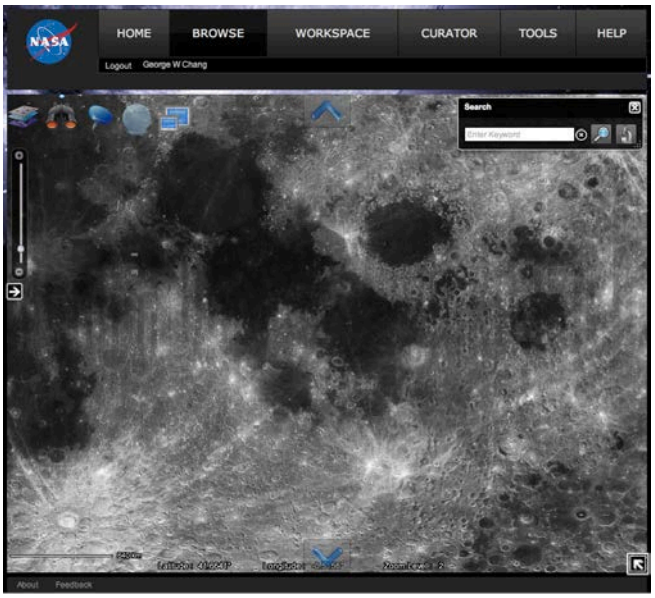

Figure 1 LMMP Portal

In addition to the web interface, we've also ported the viewing functionality to mobile devices, such as Apple's iPad (see Figure 2). This app allows users to view and layer lunar map images from the Internet. Both the web client and the mobile applications are made possible with the techniques outlined in this paper.

Figure 2 LMMP iPad App

As previously mentioned, LMMP also consolidates data acquired from missions that span over five decades, from the early Apollo missions to the latest and high definition cameras aboard the LRO and LCROSS spacecraft. These data products are mapped to the lunar surface through a common coordinate system and LMMP is working to register these data products with the to-be-established LRO control network [1]. This will facilitate more accurate information layering such as the potassium concentration map in Figure 5.

Additional examples of some of the data that LMMP serves include:

Figure 3 shows a complete lunar map composed of many regional images collected by the Ultraviolet/Visible camera (UVVis) aboard the Clementine spacecraft in 1994. The image size is 92160x46080 (4 gigapixels) compressed with JPEG2000 to 200 megabytes with a physical resolution of approximately 100 meters/pixel.

Figure 4 shows a composite image consisting of multiple images taken by Apollo 15 Metric Camera imaging system as it orbited the moon [2]. The size of the image is 49152x33496 (1.6 gigapixels) as a 3.3 gigabyte TIFF image. The physical resolution of the image is approximately 10 meters/pixel.

Figure 5 shows a coarse map of the potassium concentration on the moon. The red regions on the image show higher concentrations of potassium while the blue shows lower concentrations. The Lunar Prospector spacecraft collected the raw data to compose this image in 1998-1999 using the onboard gamma ray spectrometer.

Figure 6 shows a close-up image taken by the narrow angle camera aboard the Lunar Reconnaissance Orbiter. The image is (only a sub-image is shown) has a resolution of 5064x52224 (264 megapixels) as a 264 megabyte TIFF image. The physical resolution of the image is approximately 1 meter/pixel. We currently have over two thousand of these images.

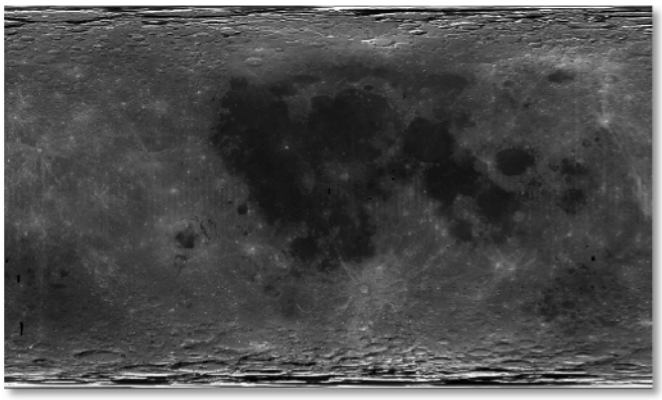

Figure 3 Global Lunar Map from Clementine

Figure 4 Apollo 15 Orbit 33 Mosaic

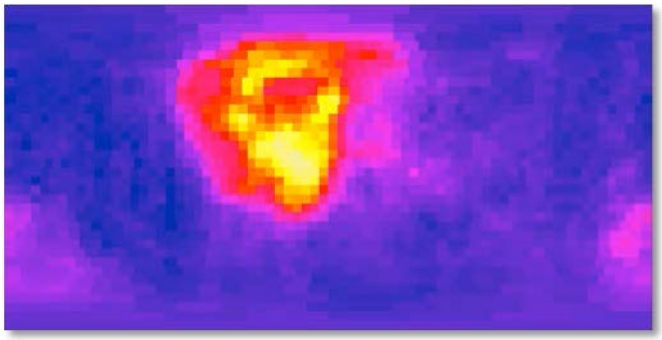

Figure 5 Potassium Concentration Map

Figure 6 LRO Narrow Angle Camera Image

These images are only a subset of the types of image that are stored in LMMP and many are of similar or larger sizes.

In a normal scenario, users view images by opening up the entire image file on the users' computer – loading the entire image into memory. However, this is not possible with images used in LMMP. For example, the image from Figure 3 would require over a thousand 30-inch monitors to show the image at full resolution. This is assuming that the memory can even be loaded into memory. A solution is required that partitions the larger image into several smaller images and facilitates on-demand retrieval of those images. Additionally, due to the rapid influx of new data, we also need to be able to process this data quickly and efficiently. We solve these issues by leveraging Amazon's cloud services, including the EC2 elastic computing infrastructure for image tiling and processing, as well as the S3 platform for data storage and retrieval. EC2 allows us to dynamically scale up and scale down our computational capacity as needed. This allows us to pay and use compute resources only when we need it. The S3 platform offers scalable storage at commodity pricing.

The goals of LMMP are two-fold. The primary goal of LMMP is to provide engineers a comprehensive data repository to support various types of analyses. Engineers involved in mission planning may use the rock and crater maps to identify regions of the moon where there are potential hazards and use the information to choose potential landing sites. Other scientists may use the latest high-resolution images to identify features of interest. In addition to processing new data quickly, LMMP also needs to maintain security so that protected and private images are only viewable by those that have permission.

The second goal is to allow the general public to view, download or analyze appropriate datasets. The system needs to accommodate large numbers of users with minimal latency to maintain expected performance. At the same time, public users need to be limited to public data and should not be able to view private or unreleased data.

In the following sections, we will discuss how we use Amazon's Cloud Computing infrastructure to solve the technical problems that allow us to satisfy our goals for the user. We will discuss the image tiling process, distributing the computation across multiple machines, and the storage of resulting tiles for on-demand retrieval.

2. Image Tiling

As Powell et. al. describe in their description of image processing of Mars planetary images, most image processing software assumes you can load the entire image into memory. [3] However, the Mars images are too large to efficiently load and manipulate in memory. The same is true for LMMP lunar surface data. So, we must break the images

down into workable sized tiles. The general idea in tiling is to slice a large image into many small images and to merge and resize until the last merge and reduce yields a reasonably sized image that depicts the entire image. The benefits of this method are threefold: 1) images are handled in small enough sizes that the computer does not need to retain the entire image in memory 2) the iterative process efficiently reuses previously computed images for future computations and 3) the task can be parallelized across many machines.

At the beginning of the algorithm, we determine the nominal tile size. The *nominal* tile size used in LMMP is 512x512 pixels. We then look at the size of the original image to determine the depth of the tiles as:

$$\textbf{zoom depth} = \textbf{abs}(\log_2(\ \textbf{max}(\textbf{width},\textbf{height})/\textbf{tilesize})) + 1$$

We chose this algorithm to ensure that at the lowest depth the image would retain as much of the original image as possible without unnecessary downscaling. For example, a 300,000x150,000 pixel image using 512x512 pixel tiles would yield a zoom depth of 10. Thus, the image tiles sliced at the lowest depth (highest resolution) would be $2^{10} = 1024$ along the longest side of the image, which in this case is the width. This calculation leads to an *initial* tile size of 300,000/1024 = 293 pixels. The *initial* tile size is then resized to the *nominal* tile size to establish the lowest level of our image-tiling pyramid. The 293x293 pixel tile is upscaled to a 512x512 pixel tile. If we had chosen a zoom depth of 9, our initial tile size would have been 586x586 pixels, which would have required downscaling and a loss of image data. If we had chosen a zoom depth of 11, our initial tile size would have been 146x146 pixels, which would yield no more image information than an initial tile size of 283x283 pixels.

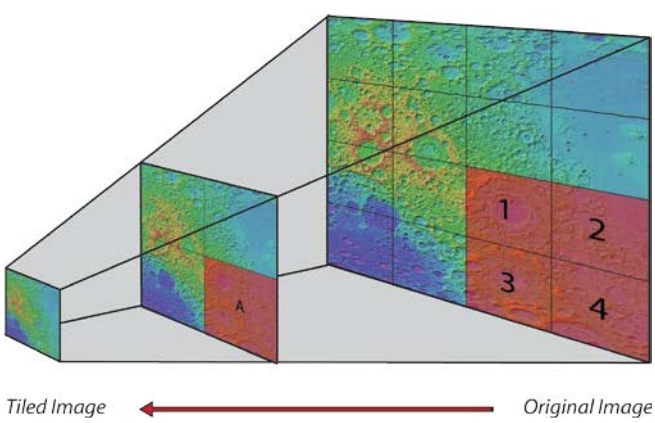

Figure 7 Image Tiling

Once we have resized the *initial* tile images to the *nominal* tile size, we can start the image pyramid generation. Figure 7 depicts a graphical representation of the tiling process. The first task is to tag a set of four tiles, aptly named NW, NE, SW, and SE. In Figure 7, these would be tiles 1, 2, 3, and 4, respectively. These four tiles would be duplicated from the original image and while retaining their original arrangement, be downscaled by a factor of 2. This resized tile is depicted as tile A in Figure 7. The resized tile is placed at index ((NW's row index)/2, (NW's column index)/2). This procedure is then performed on the next set of four tiles in the original image until all image tiles in the original image have been duplicated, resized, and positioned in the new image. If there are missing tiles near the boundaries of the original image, blank tiles are used as part of the four set tiles. Once the original image has been converted into the resized image, the process starts again with the resized image becoming the original image in the next iteration. This continues until the resized image fits on just one *nominal* tile.

3. Parallel Computing Approach

Because the tile sets at each zoom depth are duplicated and manipulated independently of the other tiles at the same zoom depth, the image tiling operation lends itself to being parallelizable across multiple computers. While there are several distributed computing software frameworks to chose from, we decided to use the Apache Hadoop framework for our implementation. [4]

The Hadoop framework, a Java-based implementation of Google's Map-Reduce algorithm, makes parallelization extremely simple and straightforward. The key concept of Map-Reduce is to take a large data set, and through two programming functions, *map* and *reduce*, produce a relatively small, yet valuable, amount of output data. The canonical case for Map-Reduce is to process web server log files. While the inputs to this problems are large files containing raw web server access information, the output are relatively smaller, usually summary statistical information about web service usage. The problem we are trying to solve is similar in that our initial input is a very large image, potentially many gigapixels in size. The output at the highest zoom depth is the same image on a 512x512 pixel tile.

The *map* step of the algorithm takes a subset of the input data and performs some computation and returns an answer. The reduce step consolidates the outputs from the map step and returns the final answer to the problem. In the web server log file example, suppose we wanted to count the number of times visitors went to the "news," "sports," or "games" section of our web site. Our input data would be the web server log files. The input to a single *map* function would be a single line in the log file. The map function would perform the string search and produce a count number for each of the keywords we were interested in. The outputs are then sorted by the keyword and forwarded to the *reduce* function, which sums the outputs of the *map* functions to yield the particular keyword. The results of all *reduce* functions yields the final answer to the problem.

The implementation for image tiling is similar. The input

data is a large image file. The image is initially sliced into tiles based on the zoom depth calculation described previously. Each image tile is then processed through a map function. The map function performs a calculation to determine where the tile image would be in the new image and its orientation relative to the other three tiles, whether it's the NW, NE, SW, or SE tile. This output is then sorted on the new location and forwarded to the reduce function. The reduce function positions the tiles and resizes the tile to produce the output tile for the next zoom depth. This process is graphically depicted in Figure 8.

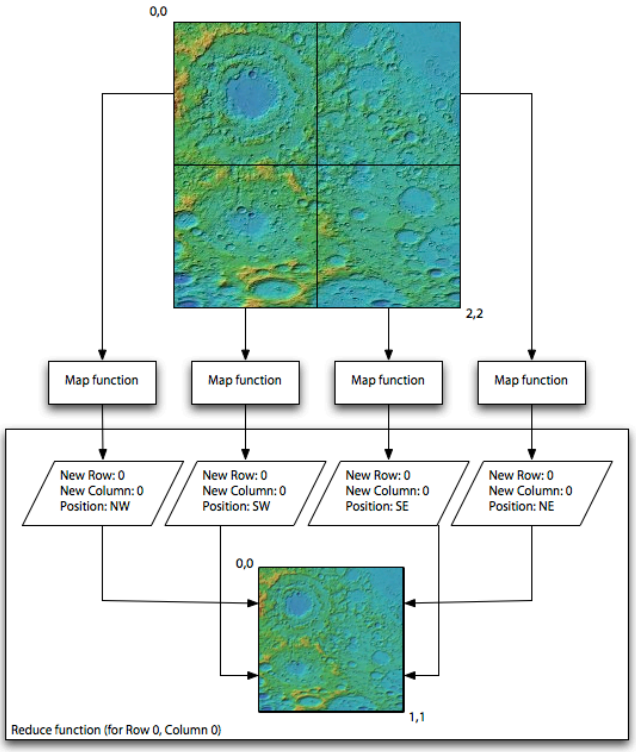

Figure 8 Map Reduce for Image Tiling

Initial Attempts

Our initial attempts at tiling high-resolution lunar images were done using a commercial tiling package on a single Windows PC. The size of the image and the processing required became readily apparent as the process took up to 10 hours and would occasionally fail due to out-of-memory problems.

We proceeded to develop a Map-Reduce implementation of image tiling and used several machines available in our lab. We used six Sun Fire x4170 machines with a heterogeneous mix of Solaris 10 and Linux, some with hard drives, and others with solid-state drives. All the machines were configured to run both *map* and *reduce* tasks, up to the limit of available CPU threads on the system. This setup worked well initially, however, it became apparent on reflection that there was a limitation to our proposed hardware configuration.

While the machines cost tens of thousands of dollars to purchase and require maintenance costs to keep the machines cool and powered on, we had never used these machines to capacity all at the same time. Because of the Hadoop framework, we only needed to specify how many CPUs to use and could let the software automatically schedule tasks to fully utilize the system. As our performance statistics later show, this local configuration produced better performance metrics on average. However, we determined for our problem we did not have real-time performance requirements, we just needed reasonable performance that could scale to the amount of data we were receiving at any given time. Owning and maintaining our own machines is a costly proposition when you can get access to machines at a fraction of the cost and at any time you may need them. Further, we needed a solution to our scale problem, which our local installation was not going to solve. After considering the costs and issues of standing up the necessary infrastructure we opted for another solution. Instead we could acquire and manage a small core group of machines for processing competition-sensitive data and rely on external machines for processing the bulk of the public domain data. The security issues associated with this choice are described more in 4.3 below.

So, we decided to look at using outside cloud computing infrastructure. At the time, just two years ago, for the feature set we wanted, there weren't a large number of choices. Our evaluation indicated that Amazon's EC2 for computations and S3 for storage and distribution would meet our requirements.

4. Cloud Implementation

Amazon EC2 is a cloud computing infrastructure that allows users to "rent" virtual machines by the hour and only pay for their usage. Amazon offers different virtual machines sizes with different speeds (Elastic Compute Units), RAM, and local storage. Depending on the service agreement, users can run anywhere from 1 to 100 or more instances for computations. [5]

In addition to providing generic virtual machines, Amazon also provides a service on top of EC2 called Elastic MapReduce, which preconfigures Hadoop on a given number of virtual machines. The inputs to Elastic MapReduce are the Java libraries that implement the *map* and *reduce* functions and data inputs from Amazon's S3 service. While this simplifies configuring and setting up Hadoop, we felt the input parameters to the service were too limiting and we also required the use of external Java libraries. We opted to configure Hadoop ourselves on generic EC2 instances.

The process to configure Hadoop on an EC2 instance requires time initially to install Hadoop, configure Java and the requisite external libraries for our task, and to develop scripts to update configuration files. However, once this machine is configured, we can use EC2's feature to save a machine image and to duplicate the machine. When we need to start a cluster, we can start up a number of virtual machines, all using the same disk image with the saved Hadoop configuration. Once the machines are up and running, we designate one machine to be the master node and execute a script which will update Hadoop's configuration file to know the IP addresses of the slave nodes.

Once our EC2 cluster is configured, we transfer the initial image file to the master node of the cluster and store it on the local storage of the virtual machine. We then run the program that will tile the image.

The program, before invoking MapReduce, will run a process to split the large image into the set of *initial* tiles. This task, which could be computationally intensive, is relatively fast compared to the subsequent merge and resize tasks. This process first reads the raw bytes needed for the tile (may need to decompress that portion of the image) and encodes it in PNG format. This PNG data is stored into Hadoop's file system (HDFS) that is automatically propagated to the slave machines. Once this initial slicing is performed, MapReduce begins.

The output from each iteration of Map-Reduce is stored inside HDFS as it is needed for the next iteration. When all zoom depths are complete, the output is written out to the file system or directly to S3.

4.1 Performance Results

For our tests, we will use 3 different configurations and compare and contrast the performance with consideration to cost.

Our first configuration will be the Hadoop cluster running on our own machines, running on 2 Sun Fire 4170's using hard drives with gigabit interconnects. These machines cost $10,000 each to purchase and require power, persistent cooling, floor space and a system administrator.

The next configuration will use 20 EC2 large instances running in a Hadoop cluster. These instances cost $0.34 per instance per hour.

Finally, our third configuration will utilize the new cluster compute instances on EC2. We will run 4 instances at a cost of $1.60 per instance per hour.

All configurations use Java 1.6 with Hadoop 0.20.2. External libraries required include Java Advanced Imaging 1.1.3 with ImageIO-Ext 1.0.8 and BigTIFF support, as well as the GDAL libraries 1.7.2 compiled with Kakadu 6.3.1. The image we will use is a 2 GB global lunar map.

	Upload Time	Processing Time	Storage Time
Configuration 1	N/A	0:37	0:12
Configuration 2	0:58	1:45	0:42
Configuration 3	0:58	0:53	0:34

Table 1 Processing Times

Upload time is the time required to upload the data for processing. In configuration 1, we already store the image data on our machines for processing. However, for new images, we receive them from the other institutions would create the images from raw data and in those cases, the upload time should be considered. Processing time is the amount of time that it takes Hadoop to run. Storage time is the amount of time required to store the data to the file system from HDFS.

4.2 Performance Analysis

Our results show that in the best case, Configuration 1 yields the best results with Configuration 3 following close behind. Configuration 2 is the slowest out of the three tests. We do not contend that files processed on local machines will perform worse than files processed on the cloud, especially if the machines are relatively new and have significant processing capability. However, we do show that in terms of a cost-benefit point of view, EC2's "rental" model offers better performance per dollar than having to purchase and maintain our own machines.

4.2 Image Storage

After the tiles have been generated, we need to make the images available to users of LMMP. LMMP runs on its own web server, which we host at JPL. However, as we expect the system to be accessible to the general public, we may encounter capacity issues. We may have problems when we have hundreds of users accessing our website, retrieving gigabytes of image data.

Just as EC2 allows us to use and pay for just the resources we use, S3 allows us to host files and pay for just the bandwidth we consume. This eliminates the need for us to purchase extra hardware and bandwidth to handle the occasional spikes in usage.

Our current implementation had some image data on S3 and others on our own servers.

4.3 Security Considerations

LMMP obtains lunar data from various spacecraft at different times. Some data are publicly released and do not require user authorization. Other data are time sensitive, such as the most recent LRO images, or they have other scientific restrictions to prevent general access. In all cases, we are also required to keep access logs to determine system usage. LMMP uses a hybrid cloud and local storage to solve this security problem.

For private data, we retain all the data and host them through our own servers. This ensures that LMMP is the sole custodian of the data and eliminates the potential security hole where a system administrator for the cloud infrastructure could potentially access the files. For public data, we store the images on Amazon's S3 storage service. From the users' point of view, the retrieval of images is completely transparent as the client application seamlessly integrates images from both sources to present to the user.

To address the usage statistic issue, we maintain logs of all accesses. For files hosted on our own system, this is relatively straightforward. Each access will be tracked in the access files. For files stored on the cloud, this may be less obvious because clients are connecting directly to cloud servers to retrieve data. We solve this by requiring users to access our servers, but only for the purpose of obtaining a signed URL to the file on S3. A signed URL is a time-constrained URL that will expire after a certain length of time. Once the client application obtains the signed URL, it then automatically uses the URL to retrieve the image.

While this process requires our servers to provide some data, the amount of bandwidth used is significantly reduced and offloaded to S3. Instead of our servers transferring the typical 100K image data, we only transmit typically 200 bytes, the amount of data needed to return the signed URL to S3.

5. CONCLUSIONS AND FUTURE WORK

In this paper we described the approaches that the Lunar Mapping and Modeling project used in solving its computation and data distribution problems through the use of cloud computing. We described an implementation of the MapReduce framework, Hadoop, and how it improved our computation times. We also described a multi-tiered data access approach that leveraged cloud storage and utilized local data in a way that provided the best data possible for the credentials of the user.

We plan to conduct further research into cloud capabilities that hold promise in improving our results. It appears that cloud computing is in its nascence and has the potential for many more capabilities. Amazon alone has accelerated their cloud feature rollouts rapidly, almost doubling their feature releases every year. 2009 had 43 feature releases of note [6]. One of those features, competitive spot pricing for unused EC2 capacity, could further improve the economics of our solution [7]. And, while Hadoop has shown to provide tremendous performance improvements, the data must still be served to the edge user in a timely manner. For mobile apps, caching information on the device is not an option; the data must be staged close to the user to make this solution practical. One related cloud feature we are interested in evaluating is CloudFront. This new Amazon capability, still in beta, delivers static and streaming content using a global network of edge locations [8]. This solution is integrated with the other Amazon facilities used by LMMP such as EC2 and S3 and shows promise for addressing edge distribution of data.

And finally, this implementation is relatively generic and can be utilized outside of this particular domain. We are also looking into offering this as a generic service to other customers.

ACKNOWLEDGEMENTS

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

[1] A Standardized Lunar Coordinate System for the Lunar Reconnaissance Orbiter, http://lunar.gsfc.nasa.gov/library/451-SCI-000958.pdf).

[2] Broxton, M.J., Moratto, Z.M., Nefian, A., Bunte, M., Robinson M.S., Preliminary Stereo Reconstruction from Apollo 15 Metric Camera Imagery, 40th Lunar and Planetary Science Conference (2009)

[3] Powell, M.W.; Rossi, R.A.; Shams, K., A Scalable Image Processing Framework for Gigapixel Mars and Other Celestial Body Images, Aerospace Conference, 2010 IEEE

[4] Hadoop MapReduce http://hadoop.apache.org/mapreduce

[5] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2

[6] "Is Amazon Winning the Cloud Race?" CloudScaling.com, http://cloudscaling.com/blog/cloud-computing/is-amazon-winning-the-cloud-race

[7] Sangho Yi; Kondo, D.; Andrzejak, A., Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud, IEEE 3rd International Conference on Cloud Computing.

[8] Amazon Cloudfront, http://aws.amzaon.com/cloudfront

BIOGRAPHY

George Chang is a *Senior Software Engineer at NASA's Jet Propulsion Laboratory (JPL). Mr. Chang developed the LMMP tiling approach and implementation, the security infrastructure as well as the LMMP mobile apps for the iPhone and iPad. Mr. Chang has a B.S. in Computer Science from Cornell University, an M.S. in Computer Science from Columbia University and is currently completing his MBA at UCLA's Anderson School of Management.*

Shan Malhotra *is a Senior System Engineer at NASA's Jet Propulsion Laboratory (JPL). Mr. Malhotra is the system architect for the LMMP Infrastructure task around which this paper is based. Mr. Malhotra has a B.S. in Physics from the University of Alberta, Canada, and an M.S. in Electrical Engineering from the University of Southern California.*

Paul Wolgast *is the Technical Group Supervisor for a group developing Deep Space Network (DSN) software systems at NASA's Jet Propulsion Laboratory (JPL). Mr. Wolgast has twenty-five years of software development experience in systems software and information technology. He began his involvement with JPL developing monitor and control software for the DSN and later led a Java Specification Request (JSR) in collaboration with a number of other companies. Mr. Wolgast has also been a contributor to software architecture initiatives at JPL.*

[3] Shams, Khawaja S.; Powell, Mark W.; Crockett, Tom M.; Norris, Jeffrey S.; Rossi, Ryan; Soderstrom, Tom; Polyphony: A Workflow Orchestration Framework for Cloud Computing, Cluster, 10th IEEE/ACM International Conference on Cloud and Grid Computing (CCGrid), 2010

[5] Pandey, S., Barker, A., Gupta, K.K., Buyya, R.,Minimizing Execution Costs when Using Globally Distributed Cloud Services, 2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)