WHITE PAPER

TITLE

LARGE VOLUME MAP GENERATION VIA GRID COMPUTING

ABSTRACT

Trekk Cross-Media has been using ArcWeb Services for several years to generate custom route maps from a residential location to a point of interest for use in direct mail campaigns. These campaigns can range in volume from 10,000 recipients to 250,000 or more. Each requires generation of a custom image.

As demand for this service has grown, the need to produce more maps per hour has also grown. Trekk has evolved the process to increase production from 500-750 map images per hour to more than 20,000 images per hour. Trekk now uses a grid computing architecture to harness the throughput of dozens of CPU’s within our corporate environment that would otherwise go unused.

This presentation will provide a technical overview of the environment created and tips on using ArcWeb Services for this process.

TREKK CROSS-MEDIA

LARGE VOLUME MAP GENERATION VIA GRID COMPUTING

TABLE OF CONTENTS

Before the rise of mass communications technology, such as radio, television, and high speed printing presses, communicating your message to prospects and customers was done on a one-to-one basis. While mass communications enabled marketers to get their message to the largest possible audience at the lowest possible cost, it also resulted in a saturated marketplace that has pushed consumer resistance to an all-time high.

Today, there is a growing movement to return to the age of one-to-one communications, and new technology is making that mass customization possible through personalized web pages, customized mass email, on-demand video, wireless devices and variable digital print. Organizations can now create customized, personalized, relevant messages based on the target person's preferences, purchase history, and areas of interest and deliver messages cross-media to the recipient's channel of choice.

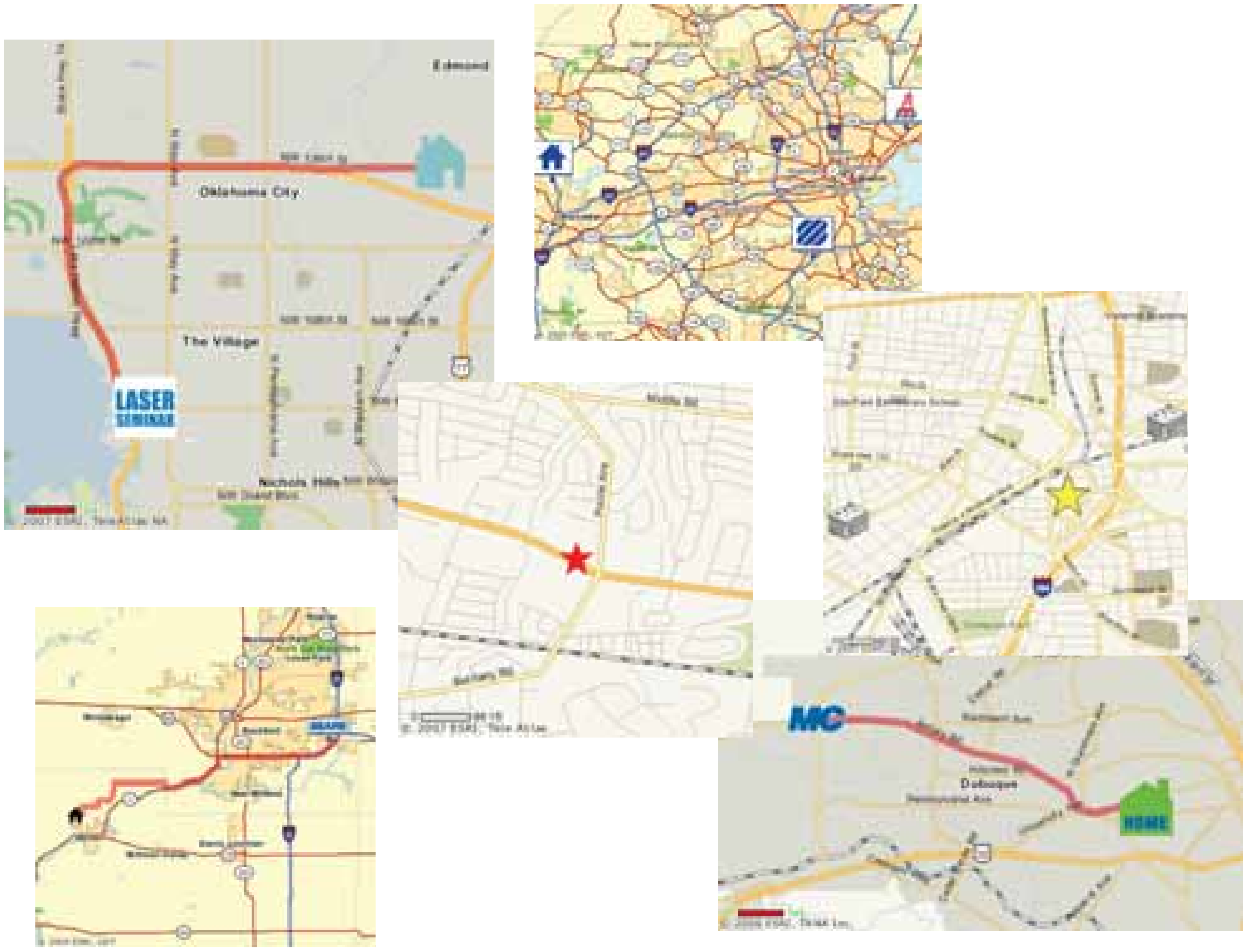
As personalized communication becomes more commonplace, people increasingly expect it. The challenge for marketing professionals is to create messages that are not only personalized but relevant. One technique is to include a custom map image in online and print communications that directs recipients to a specific location, such as the closest retail outlet for a particular brand.

In the past, these images have been variable based on the general area in which a person lives. With the advent of several generations of web based mapping tools, consumers have come to expect more. It is now possible to generate custom route maps that detail the traffic route between a consumer's residence and the local point of interest. These maps, included in highly personalized, digitally printed communications, generally result in increased response rates, demonstrating the power of one-to-one communications.

It is easy enough to use one of several online mapping tools to generate a custom route map. It is even possible to manually generate dozens of maps for use in custom communications. The trick is in generating thousands or hundreds of thousands of map images for large-scale custom communications.

Trekk has developed a solution that can produce several varieties of map image types, is tolerant of faults that arise out of the process and has scaled to several hundred thousand maps per day.

When we first started the development process, we knew we needed an image type that will work with our digital print layout software. We also knew we wanted to use the map styles that consumers were familiar with from web-based route maps vendors. Based our experience with web site development, we investigated the tool vendors that help produce online systems. These included Microsoft, MapInfo and ESRI.



We were also concerned with the cost associated with map production. Each vendor has different fees associated with licensing of the map generation tools, as well as the underlying map layer data. Since we were experimenting with route maps on a small scale as a proof of concept, we were immediately drawn to Web Service (WS) based offerings from Microsoft and ESRI. They allowed us to experiment at the lowest possible cost and removed the need to install large mapping databases in our facility. Subsequent to our experimental phase, other vendors also began to offer WS based toolkits.

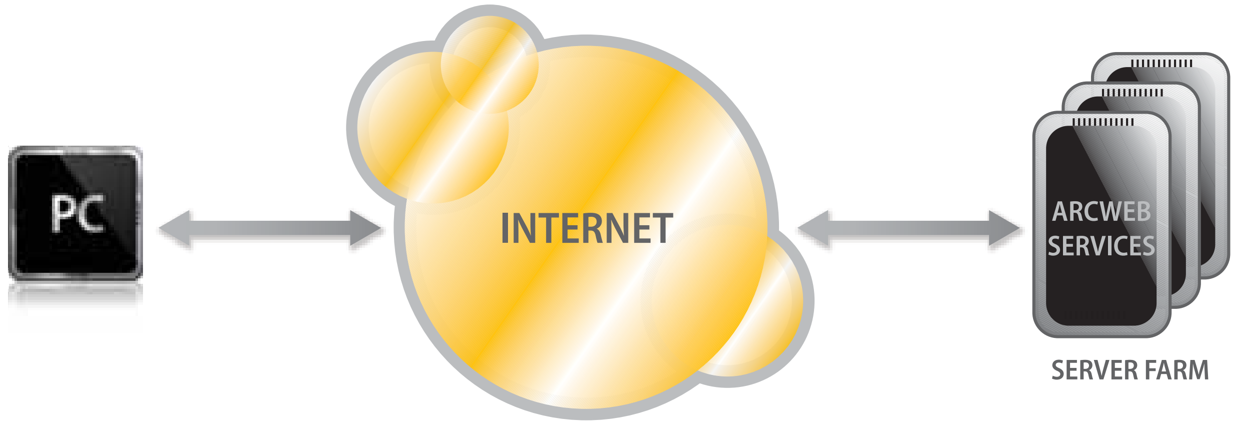
Based on our experiments, we decided to work with the ESRI ArcWeb Services product. The deciding factors were the completeness and simplicity of the documented WS interfaces and the low entry price and smaller unit of usage packages available. From our experiments, we developed an executable package that allowed us to run large volumes of map images for our projects.

We ran several projects of 10,000 and 20,000 images per batch. Our initial configuration was a command line executable .Net application connecting across the Internet to the ArcWeb Services web server farm. This program would perform six steps:

1) Read a row from the recipient database
2) Call the ArcWeb service to geocode the address of the recipient
3) Find the nearest point-of-interest from a preconfigured dataset on the ArcWeb service servers
4) Call the ArcWeb service to create and map a route from the residence to the location of interest, complete with custom icons for each
5) Download the image file and update the recipient database with the image reference

We originally chose to create jpg images to be used in our print pieces. In print, we would rather use a vector image to create high-resolution output for digital printing. But ArcWeb Services, like most every other similar product, is geared toward the creation of web images. We have overcome this problem through trial and error and now typically create PNG8 files for print production. However, we would still prefer an EPS or similar output files type to reduce the size of each image. While ESRI and others now offer Flash compatible vector files, we have not yet found these suitable for print.

SINGLE CPU SINGLE TASK



This process involved a single program running on a single CPU executing as a single task. Our initial runs resulted in throughput of about 600 to 800 images per hour. For a run of 10,000 images, more than fifteen hours of production time was required. We also learned about the variety of ways the process can fail. The failure modes are numerous and include network performance and reliability issues, bad addresses that cannot be geocoded, maintaining session states with the web farm over the series of calls, memory management issues for large datasets, and a host of others. In fact, a huge chunk of development time was spent identifying and managing error-generating conditions. (Anyone developing Service Oriented Architecture (SOA) and Web Services based systems should keep this in mind.)

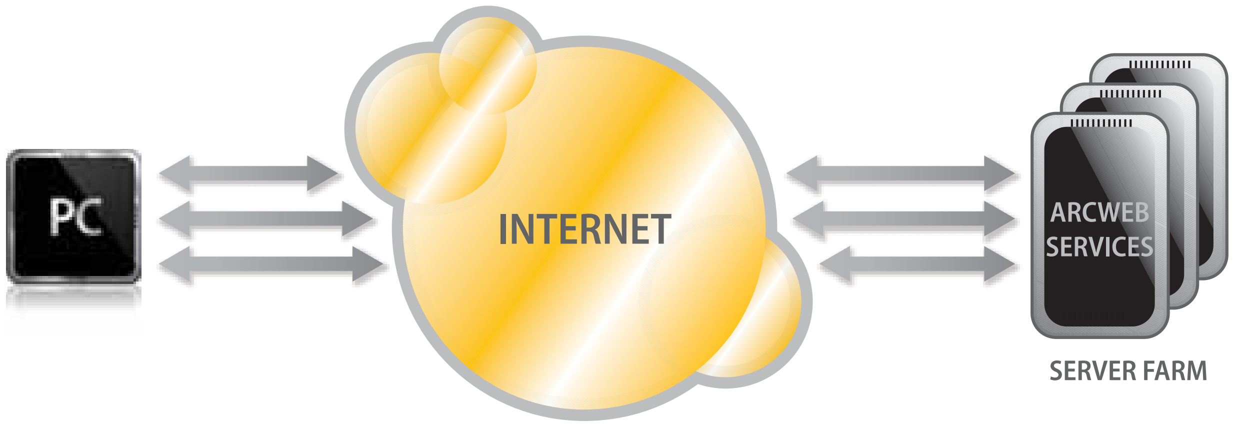
We profiled the application to identify bottlenecks and scaling constraints. For this project to be commercially viable, we needed throughput of at least 10,000 images per hour. At this point, we were using ArcWeb Services as a low-cost version of ArcGIS. We felt that we could scale this system by creating a dedicated database server and execution engine. We could then beef up the hardware to get the performance we needed.

But we noticed that this was not a CPU nor memory constrained application; it was network bound. The two longest steps were waiting for the web farm to respond to calls and downloading the image file. We used the services of the ESRI technical staff to help determine the best course of action to scale the performance. Rather than build a large mapping database ourselves, ESRI already had built one in the form of the ArcWeb Services web server farm. Our problem was how to increase the request rate from our application.

The next version of the application was designed to provide more requests per minute to the web farm. We evolved it from a single-tasking executable into a multi-tasking, multi-threading application. By using the .Net threading libraries, we created an application that spawns multiple tasks simultaneously and then waits for their completion. Each task was an independent execution sequence similar to the original application. Each task executes in a separate thread of execution and notifies the master application when completed. The master application then launches another thread until all rows of the database have been exhausted.

We created the application to allow the modification of the number of simultaneous threads managed to test for the optimal configuration. We also allowed multiple copies of the executable to manage different sets of the database to create as many simultaneous requests as possible.

SINGLE CPU MULTIPLE TASKS



We found that by simply multiplying the number of tasks and requests from a single CPU (and single network connection) the images per hour generated did not increase linearly. In fact, it seemed to max out rather quickly. We found that with this configuration we were able to produce 1,500 to 2,000 map images per hour.

We ran several projects with 50,000 to 200,000 images with this configuration. Even with the increase in performance, the production time was measured in days. And while the code was developed to detect and manage error conditions, a significant amount of time was required to supervise and rerun failures. We still needed to increase performance and increase error resilience.

During testing we found that running multiple versions of the code on separate systems increased the throughput versus running multiple versions on a single system. Since the application is not CPU or memory bound, the bottleneck was LAN and Internet connections. We felt that if could scale the application over a number of systems with separate IP addresses and network connections, we might be able to scale the system to the performance we needed.

A type of system architecture called grid computing allows applications to be widely distributed across computing resources. Grid architectures are typically used to help solve scientific problems. Many of these problems require substantial CPU resources. By distributing portions of the problem across systems, many CPUs in addition to the master system can be harnessed. The individual systems then return the solution of their small piece back to the master system. The master system assembles all of the parts resulting in a complete solution.

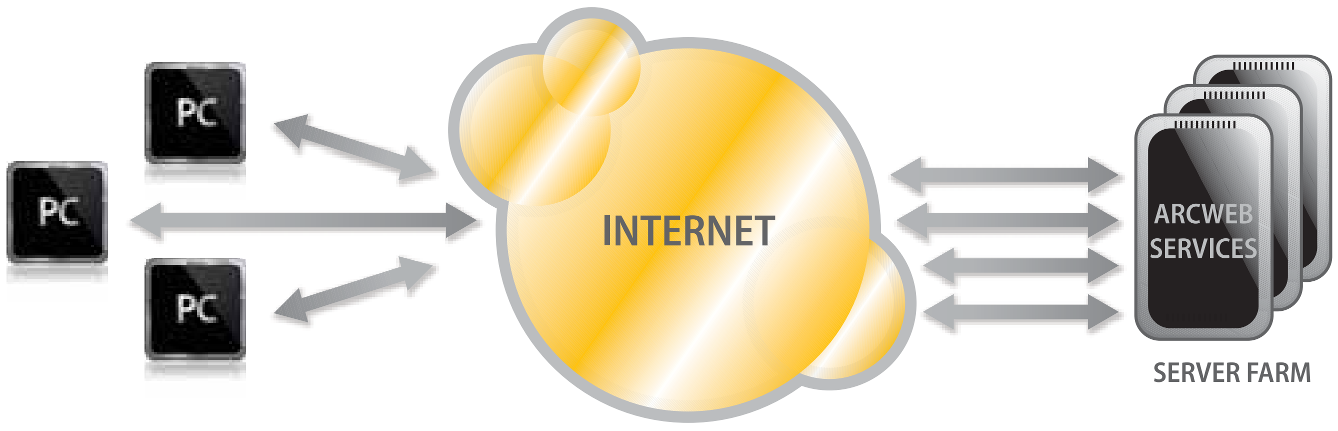
We knew that our problem did not require substantial CPU resources, but it did require numerous network connections to the web farm. The challenge to such architecture is building the code to distribute and manage all of the work and deal with failures and errors, as well as to collect the results back in a central location. To help with this problem, we identified grid computing vendor Digipede, which offers a Windows-based toolset.

The advantage of a Windows solution is that we could harness the dozens of Windows systems in our internal network. A small Digipede client application is installed on each system. And then in the background, a master system sends work to available agents. And when the systems are not otherwise busy, (most have no one using them 128 out of 168 hours per week), they can work on the distributed problem.

Such grid computing systems have been used commercially in scientific areas and to harness the systems connected to the Internet. Several volunteer Internet Grid projects that are popular with the public are as diverse as the Compute Against Cancer projects, the Climate Prediction project, and the SETI@Home project. These projects gather the computing resources of millions of volunteers to help solve problems like 3-D analysis of protein molecules, computing weather models, and searching for extra-terrestrial life in radio signals.

We felt we could use that same grid approach to create high-volume map image generation system using internal system resources that would otherwise go unused. We set up an experiment with the Team Edition of the Digipede Network. We rebuilt our mapping engine to work as a Windows executable that was assigned one row of the dataset. We configured the Digipede Controller to create a distributed task for each row in the dataset. We also configured it to collect the downloaded image file from the agent system with each task competed. The images were all saved in a central location.

MULTIPLE CPU's SINGLE TASK



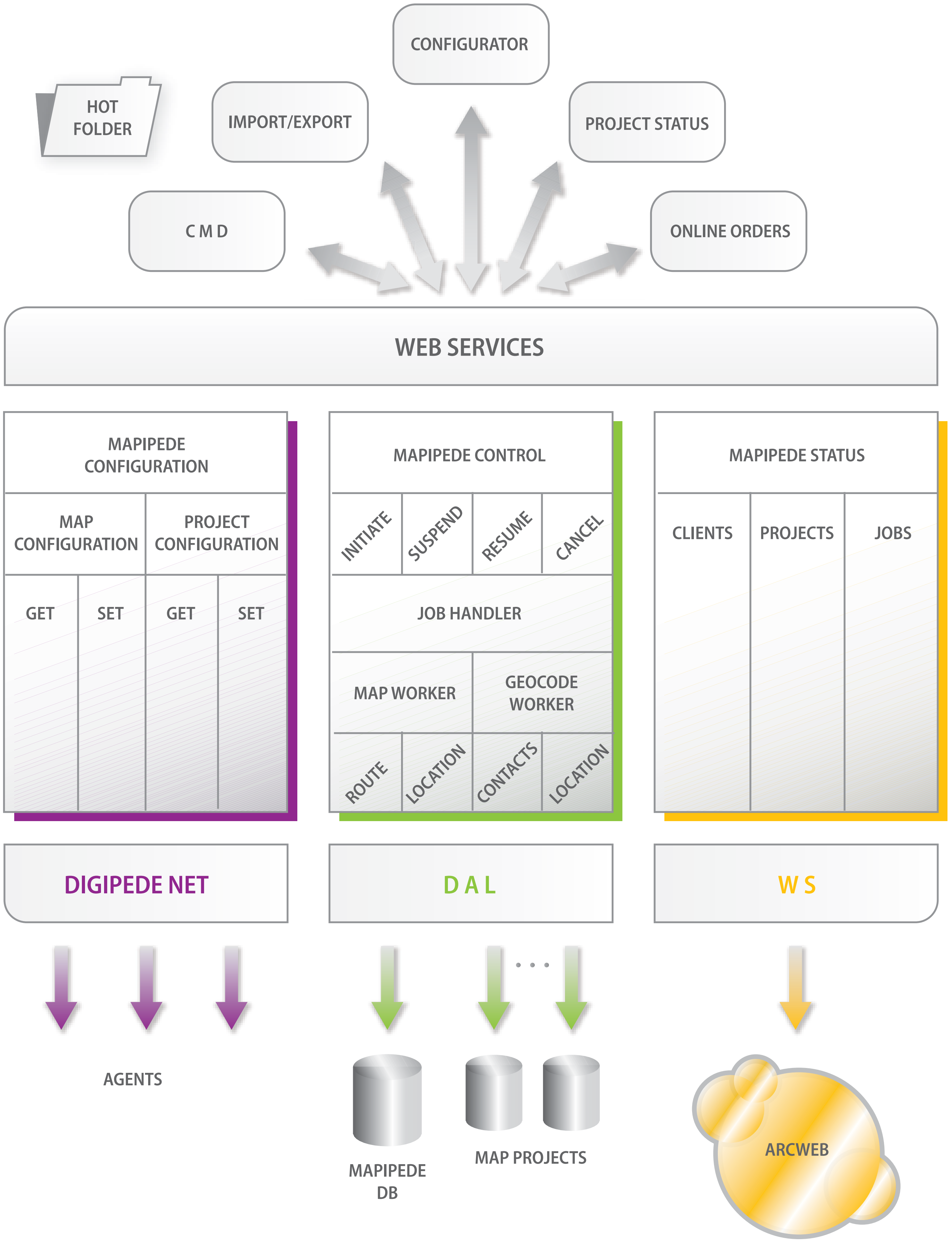
The results were impressive. Our first project run almost doubled our previous throughput. We excitedly configured our network with 20 agent machines. After more experimentation, we began to run jobs at the rate of around 5,000 map images per hour. While still shy of our goal of 10,000 per hour we knew we were on the right track.

The support staffs at ESRI and Digipede were very helpful and offered a lot of suggestions. One issue we had to overcome was maintaining an authenticated session with the ArcWeb server farm. Because our network connection to the Internet passes through a firewall, it is possible that a sequence of calls from any machine can get translated into different IP addresses by the firewall. The web farm sees this as a different client and the session token fails. We eventually configured each task to make an initial call to create a new token. The security tokens we create are short lived but for the series of calls from a single task it seemed to work well versus sharing login credentials across all of the tasks.

The Digipede Network can handle fault conditions well and can reassign the same record again when failures occur. But to properly create software that takes advantage of this, we needed to migrate our code from a simple executable distributed across the network to an application that works with the Digipede API directly.

It was also suggested by support staff that we should see performance improvements because the API implements a technique known as .Net Reflection. Essentially that means instead of copying an executable from a server to the agent systems file system, Reflection can help create an assembly of code and have the network copy it from the main application to the agent system memory. From a programming standpoint the assembly appears like any other function or method call. When the agent is done executing the task, the results appear back in the main application as returned object properties.

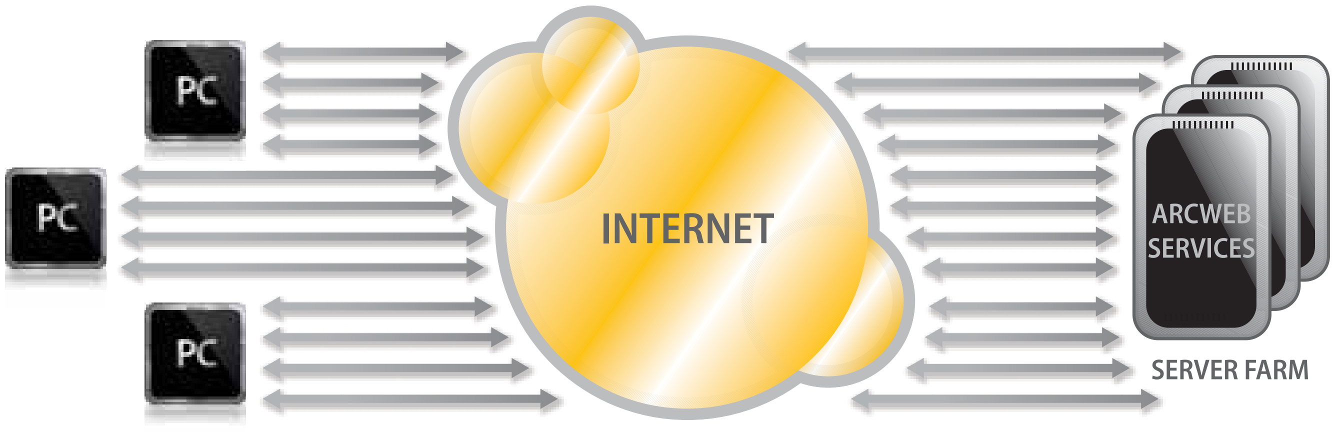
Our original goals where to process 10,000 maps per hour and to start the process, walk away and return when it is completed. We rebuilt the application using a design of layers of services. The lowest layer accesses the database, the remote ArcWeb Service interfaces, and the Digipede services.


CONFIGURATOR
HOT FOLDER
IMPORT/EXPORT
PROJECT STATUS
C M D
ONLINE ORDERS
WEB SERVICES
MAPIPEDE CONFIGURATION
MAP CONFIGURATION
PROJECT CONFIGURATION
GET
SET
GET
SET
MAPIPEDE CONTROL
INITIATE
SUSPEND
RESUME
CANCEL
JOB HANDLER
MAP WORKER
GEOCODE WORKER
ROUTE
LOCATION
CONTACTS
LOCATION
MAPIPEDE STATUS
CLIENTS
PROJECTS
JOBS
DIGIPEDE NET
D A L
W S
AGENTS
MAPIPEDE DB
MAP PROJECTS
ARCWEB

The middle layer implements business logic and processing rules. This layer is then exposed as a set of Web Services. The upper layers implement user interfaces (UI) for configuring, running and monitoring jobs. These UI applications are typically built as web applications. But we have already used the system to implement automated job requests from remote systems.

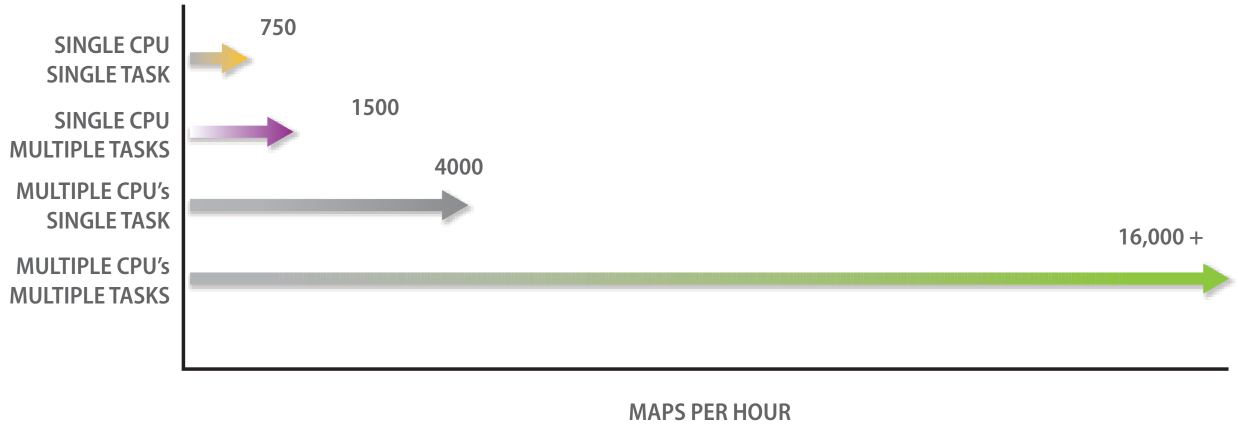
Again we created an experimental application, which required considerable manual configuration, to test the scaling and fault tolerance features. We were thrilled with the results. Not only did we see performance improvement from the direct use of the grid API, but we configured each agent to handle more than one task at a time. We began seeing jobs run at a rate more than 16,000 images per hour.

MULTIPLE CPU's MULTIPLE TASKS



With that kind of performance, we began to further analyze the different types of problems that can occur in the process. These can be categorized in to two areas: system failures and data failures. System failures can be caused by network issues, database access issues, Internet connectivity issues, and issues within the web farm. Data issues can be cause by addresses that cannot be geocoded, problems with associated point of interest assignments, and attempts to create routes that are not supported by the GIS street data.

A lot of work has gone into preprocessing of the data to make sure errors do not crop up during production. We have configured the code to retry those tasks that generate system failure errors, and we mark those that generate data failure errors so we can attempt to correct the problem. From the beginning of this project up until know, we have reduced the map generation failure rate from about 5% down to less than 0.1%. We have also increased throughput from 800 to 20,000 maps per hour.



CONCLUSION

The paper has tried to demonstrate the utilization of a unique combination of technologies to produce a commercially viable service for our clients. The project started almost three years ago. In the process, we have evolved through four major versions and architectures of the software. We were confident that the use of ESRI web services based offerings would allow us to scale this system to make it an economically viable service. And with the application of a grid architecture toolkit, we were able to accomplish our goals.

REFERENCES

http://www.esri.com/
http://www.digipede.com/products/case_trekk.html
http://setiathome.ssl.berkeley.edu/
http://www.computeagainstcancer.org/
http://www.climateprediction.net/

ABOUT THE AUTHOR

Jeffrey Stewart
VP Technical Services
Trekk Cross-Media
134 N Main St
Rockford, IL 61101
815.262.7252
www.trekk.com
stew@trekk.com