IN-DATABASE IMAGE PROCESSING IN ORACLE SPATIAL GEORASTER

Qingyun (Jeffrey) Xie, Senior Software Development Manager
Fengting Chen, Principal Member of Technical Staff
Zhihai Zhang, Principal Member of Technical Staff
Ivan Lucena, Principal Member of Technical Staff
Oracle Corporation
1 Oracle Drive, Nashua, NH 03062
qingyun.xie@oracle.com
fengting.chen@oracle.com
zhihai.zhang@oracle.com
ivan.lucena@oracle.com

ABSTRACT

Geospatial images are big data. Geospatial image processing is data intensive. Oracle Spatial GeoRaster enhances the Oracle enterprise database to natively store and manage geospatial imagery, which effectively solves the data management and scalability problems. However, with the data volume growing exponentially, real-time or near real-time image processing and database query become more challenging. This paper describes the implementation strategy of the in-database image processing engine of Oracle Spatial GeoRaster and its performance benefits. First, it not only enhances the database with advanced query capabilities, such as analytical queries and queries with image aggregation or mosaicking, but also enables massive image processing inside the database. This significantly enhances the GeoRaster data management and manipulation itself. Second, performance is the key driver behind the strategy and it has three major features to provide greater performance. The first feature is it moves the image processing closer to the images instead of moving the images out of the database to the processing. This helps achieve greater performance by avoiding data movement. The second feature is parallel processing. We parallelize some of the processing to improve performance. The third feature is concurrent processing. User can leverage the power of computer clusters and the optimized load balancing to concurrently process numerous images. The GeoRaster image processing engine supports large-scale image rectification, image appending, Virtual Mosaic, and NDVI computation among others. This paper presents some performance test results. The functionalities and performance results demonstrate that this in-database image processing engine not only dramatically improves spatial query and processing capability of large-scale image databases, but also effectively solves some of the biggest performance challenges.

KEYWORDS: image, raster, database, image processing, parallel processing, concurrent processing, management

INTRODUCTION

For geospatial image and raster data archiving and management, enterprise RDBMS technologies have been widely used as the foundation. Over the past decade, several products including GeoRaster, RasDaMan and ArcSDE, and most recently, PostGIS have demonstrated this database technology (Baumann, 2001. ESRI, 2005. Oracle, 2004. PostGIS, 2012). These products differ in their design and implementation approaches but they have at least one feature in common, that is store image and raster data inside an RDBMS database, which in turn provides great manageability and proven scalability.

Oracle Spatial GeoRaster is a large-scale geospatial image management system and platform built inside the Oracle enterprise database server. It enables users to natively store and manage massive geospatial imagery inside the Oracle database. Oracle Spatial GeoRaster is unique because it takes the database-centric approach by providing a single native data type SDO_GEORASTER to encapsulate both metadata and cell data of rasters. It provides native indexing and query capabilities as well as an internal processing engine (Xie, 2008a. Xie, 2008b).

Once a large-scale enterprise RDBMS based image database is built, many advanced and highly efficient desktop systems can connect to it to retrieve the imagery and raster data out of the database and process them in the client or another server. Such systems include well-known ERDAS Imagine, PCI Geomatica, ArcGIS, Manifold System, QGIS/GRASS, to name a few. However, moving the data between the database and the processing engine is costly given the speed and bandwidth limitations of computer networks. Geospatial images are big data and geospatial image processing is data intensive. Real-time or near real-time performance should be a key consideration at the beginning of the design of any such modern image management solutions.

On the other hand, with image data growing at ever increasing rates, relying purely on desktop image processing systems is not enough for various performance and scalability reasons. Advanced capabilities, such as more sophisticated queries and aggregation and some image transformation and processing, should become part of such database management systems.

Therefore, we takes the enterprise database-centric approach for both data management and data processing. This paper presents one of the central components of this database-centric approach: the processing engine built completely inside the database. Part of this processing engine is image processing, which we call the In-database Image Processing. This paper describes the implementation strategy of this in-database image processing engine of Oracle Spatial GeoRaster and the benefits. We focus on its functionalities and its performance features.

We also present a series of tests on the newly implemented in-database mosaicking and pyramiding functions on a 24-CPU machine. This machine is a x4170 M2 Sun Server, which is a x86 64-bit Linux machine and has 24 Intel(R) Xeon(R) 3.07GHz CPUs and 144GB memory. The storage has 8 SAS disks with a total size of 4TB. The disks are raided into a RAID5 volume. We used one logical volume of it for the test database. The operation system is Red Hat Linux 2.6.18-308.4.1.0.1.el5.

IN-DATABASE PROCESSING

In-database processing refers to the integration of data processing functionalities into the databases or data warehouses. The basic idea is to eliminate the overhead of moving large data sets from the enterprise databases to separate processing and analytical software applications.

An in-database processing and analytics approach is much faster, more efficient, and more secure than traditional approaches. In-database analytics delivers immediate performance, scalability and security improvements because data never leaves the database until results are filtered and processed (Das, 2010).

In-database processing is performed and promoted as a feature by many of the major database and data warehousing vendors, including Oracle, IBM, Teradata, Netezza, Greenplum and Aster Data Systems (Grimes, 2008. Berger, 2009). For example, Oracle Data Mining and Oracle R Enterprise are in-database data analysis engines. Coupled with the power of SQL, they eliminate data movement and duplication, maintain security and minimize latency time from raw data to valuable information.

In-database processing has been successfully used in many high-throughput and mission-critical applications. The success of this approach and its applications inspired us to consider the same strategy for massive image processing inside Oracle Spatial GeoRaster.

As we mentioned in the introduction, geospatial imagery and raster data are big data. A typical geoimage database has tens or hundreds of terabytes of data. It can easily grow to petabytes or even more data. Data has “weight” and geospatial image and raster data sets are particularly “heavy”. Given that the processing and analysis are data intensive, data locality should always be an important factor in our design and implementation strategy. So we conclude that building an in-database image processing and raster analytics engine should be a good strategy. It moves the data processing closer to the data instead of moving the data to the processing, which helps achieve better performance by overcoming the bottleneck of computer networks. It also improves scalability and security.

IMAGE PROCESSING IN GEORASTER

Unlike file systems that people are most familiar with, an RDBMS database looks more or less like a black box data storage system to some users. Particularly for geospatial image management, RDBMS systems might be incorrectly considered hard to understand, hard to use and infeasible for massive image processing workflows. While this needs broader discussions beyond this paper, we think behind these concerns there are two key questions of this in-database approach, which we need to address. The first one is whether or not the database is scalable enough to handle huge data throughput and efficiently support massive image management. The second question is whether or not we can efficiently implement massive image processing capabilities inside the database. For the first question, we conducted intensive tests and proved that GeoRaster is robust in handling virtually unlimited image data sets and can scale to large number of concurrent users (Xie, 2006). Image processing typically requires high speed disk I/O and efficient memory usage. From the beginning of this project, we started with building a robust I/O and memory management infrastructure inside the Oracle database. On top of this infrastructure, we successfully implemented many image data manipulation operations in the Oracle 10g and 11g releases (GeoRaster was first released with Oracle 10gR1 database in 2004). Those operations can handle any image size and any number of images. In this paper, we describe the more advanced image processing engine we have built in the last few years and its performance features.

Image Processing Functions

Image and raster data processing and analysis involve a large set of operations, such as radiometric and geometric corrections, large-scale image transformation and mosaicking, image enhancement, pattern recognition, raster map algebra, terrain modeling, geostatistic analysis, to name a few. While it's doable we think it's not necessary to implement all image processing functionalities inside the database. Instead, we use two major criteria in selecting what to implement and setting priorities. First, those that are required by database management or dramatically improve data manipulations, such as image updates and queries that require some image processing or aggregation, should have the highest priority. Second, they should improve dramatically the performance of massive image processing or complement traditional remote sensing and GIS applications and solutions. For example, as a result of such functions, third party solutions can benefit greatly in performance from this image processing engine by pushing some basic data processing and filtering operations into the database so that less data is retrieved and transported into the client for further processing and analysis.

With these considerations, we implemented GCP georeferencing, reprojection, rectification, orthorectification, image clipping, pyramiding, scaling, color stretching, masking, image appending, bands merging, large-scale advanced image mosaicking, and virtual mosaic support. We also designed and implemented raster algebra. The GeoRaster raster algebra is an extension to the PL/SQL language, which enables very fast cell value-based conditional queries and updates, raster data analysis, cartographic modeling, and supports image segmentation, NDVI computation and Tasseled Cap Transformation. Raster algebra is described in another paper (Xie, 2012). All those image processing functionalities are completely implemented and run inside the Oracle database server.

To illustrate the image processing operations we describe the GeoRaster mosaicking capability, compare its performance with a file system based mosaicking, and demonstrate the implementation and benefits of parallel mosaicking and concurrent mosaicking in more details in this paper.

Large-Scale Mosaicking

With imaging resolution and acquisition frequency ever increasing, the size of each image increases while the size of the area it covers decreases. This results in larger databases with larger number of images for the same area. Mostly, an image database would store both raw images and preprocessed imagery as is. Very often, applications require such images to be mosaicked for various applications and particularly for query performance reasons. If a thick client or a desktop application is used to mosaick the images, all of them would have to be retrieved from the database and shipped to the client for the mosaicking process, and then the resulting mosaic would be stored back into the database. Such data shipment is a typical database round trip and is very expensive given the volume of the images. So, it makes sense to simply mosaick the image inside the database and store the resulting mosaic directly where the source images are stored. Mosaicking capability easily falls into our selection criteria.

The newly implemented GeoRaster large-scale mosaicking function mosaicSubset allows image gaps and overlaps. It can handle both rectified and unrectified source images. It supports internal reprojection or rectification, many common point rules for dealing with overlapping pixels, and simple color balancing. You can also mosaick at a certain pyramid level. This mosaicking process results in a single GeoRaster object, which is called a physical mosaic as opposed to virtual mosaic. It is parallelized. Users can also run multiple mosaicking tasks concurrently. This capability enables massive image mosaicking processing right inside the database.

Virtual Mosaic Support

While physical mosaics are critical and provide great performance for many applications, some time mosaicking a collection of images into a single physical mosaic is not necessary or desirable. For example, you might not have enough disk space for storing the mosaic separately or you simply want to save disk space. Another example is if you do not want to keep two identical copies of the same data set but prefer to have the original data set stored as is, such as a DEM data set, yet you want to query over this data set seamlessly. Yet another example is if you want to apply different processing and mosaicking rules for the same region when mosaicking the source images. In such cases, instead of mosaicking a set of GeoRaster images into one large GeoRaster image and storing it in a GeoRaster table, you can create a virtual mosaic. The implementation of virtual mosaic falls into both our selection criteria, particularly the first one.

In GeoRaster, we define a virtual mosaic as any large collection of georeferenced GeoRaster objects (images), rectified or unrectified, from one or more GeoRaster tables or views that is treated as if it is a single GeoRaster object (image). A virtual mosaic can contain unlimited number of images, and a whole GeoRaster database can be treated as a virtual mosaic. You issue a single call to query the virtual mosaic based on area-of-interest (that is, subsetting or cropping), and you can request the cropped images to be in different coordinate system with different resolutions. Just like with the physical mosaicking process, on-the-fly transformations with resampling and mosaicking with common point rules, based on user requests, are done internally and automatically during the query processes. More specifically, there are three very easy yet very flexible and powerful ways to define a virtual mosaic:

(1) As a GeoRaster table or a list of GeoRaster tables
(2) As a database view with a GeoRaster column
(3) As a SQL query statement (a cursor) that results in a collection of GeoRaster objects

These methods allow you quickly define any collections of images in the same database as virtual mosaics. You don't need to create a new file to describe or contain the images and their full path file names, which are typically required by a file system based solution. After a virtual mosaic is defined, you issue a single call to the getMosaicSubset function, which directly returns a single mosaicked image precisely covering your query window or area-of-interest for display and other applications. This enables numerous on-the-fly queries over a virtual mosaic no matter how many images are in the virtual mosaic. You can also call the large-scale mosaicking function mosaicSubset to perform the queries and mosaicking and store the result in the database as a persistent GeoRaster object.

Essentially, virtual mosaic is used not only as a large-scale mosaicking process but more importantly as a database query and search engine. It can be considered as an image serving tool as well. Smaller source images are stored as is. But by using the virtual mosaic definitions and the two functions, the engine enables database queries and sophisticated spatial, temporal and associative search to filter out the relevant images and then automatically aggregates, resamples, mosaicks and crops them to be returned as a single image result.

Mosaicking Performance

We also achieved great performance for the in-database image processing functionalities. On the 24-CPU machine we described in the introduction, we conducted some performance comparison between a file-based mosaicking function and GeoRaster's mosaicking function mosaicSubset. GDAL is a great ETL tool for transforming different image formats, including support of the GeoRaster format for very fast importing and exporting of images stored in GeoRaster. It also has some data processing utilities, including a reprojection and mosaicking function called gdalwarp (GDAL, 2012). We use the file based gdalwarp to do the comparison.

The source testing data set includes 31 Landsat 5 Level 1T TM images, which are obtained from the U.S. Geological Survey. They cover the northern California area. The source images have 7 bands and around 7251 rows and 8121 columns of pixels each. The cell depth of the image is 8 bits and the interleaving of the pixels is BSQ. The total size of the image set is 11.9 GB. Among the 31 source images, 15 of them are in WGS 84 / UTM zone 11N projection (SRID 32611) and 16 of them are in WGS 84 / UTM zone 10N projection (SRID 32610), thus the mosaicking processes will do reprojection on about half of the images. The output mosaic image has the same cell depth and interleaving as the source images. The mosaicked image has 40523 rows, 27378 columns, 7 bands and SRID of 32610. It is about 7.7GB in size. Figure 1 is an overview of the resulting mosaic.

GeoRaster's mosaicking function has more advanced features but we made sure gdalwarp and mosaicSubset run the same functions and achieve the same results. We tested on both single-thread and parallelized mosaicking. In the parallelized case, multithread option (-multi) is used in the GDAL gdalwarp command, though there is no option to specify the parallel degree. Our mosaicSubset used parallel degree 8 in the SQL hint. The parallel gdal command and georaster script are as follows:

```
-- GDAL command:
-- required by gdal, mosaic.vrt is a file predefined to describe all 31 landsat images
gdalwarp  -t_srs EPSG:32610 -srcnodata 0 -dstnodata 0 -r near –multi
    /landsat/mosaic.vrt /landsat/mosaic.tif

-- GeoRaster mosaicking script:
-- select all 31 images that are stored in the tm_images georaster table
```

```
stmt := 'select grobj from tm_images where id > 0 order by id';
open cur for stmt;
-- mostly use default parameters to mosaick. this ensures the same functionality as gdalwarp is called
sdo_geor_aggr.mosaicsubset(cur, null, 32610, null, null, null, null, null, null, null,null,
    'nodata=true', 'blocksize=(512, 512, 7)', gr, null, 'parallel=8');
```

The testing result is shown in table 1. As it shows, the GeoRaster mosaicking function without parallelism is 2.3 times faster than GDAL's mosaicking function. If both run in parallelism, the GeoRaster mosaicking function with DOP = 8 is 11.3 times faster than GDAL's multi-threaded mosaicking function.

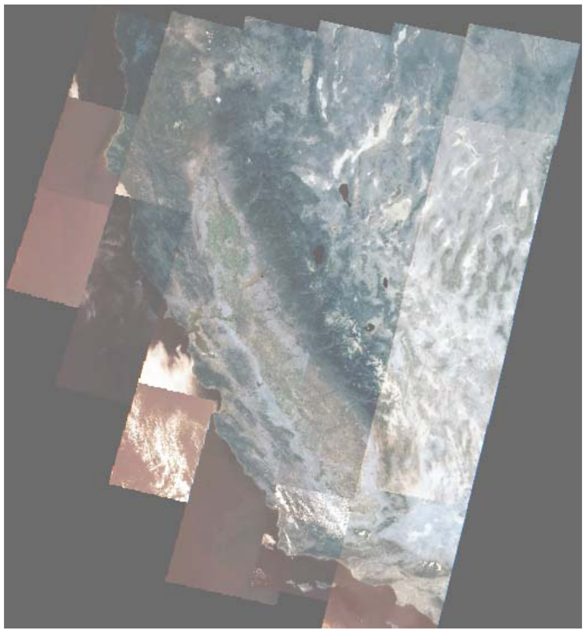

Figure 1. Overview of the Mosaic of 31 Landsat TM Images.
(Image Courtesy of the U.S. Geological Survey)

Table 1. GDAL and GeoRaster Image Mosaicking Time in Minutes

	non-parallel	**parallel**
GDAL mosaic	43.12	42.1
GeoRaster mosaic	18.65	3.73

In summary, it's feasible to implement any complicated image processing algorithms inside the database and great performance can be achieved. In-database image processing functionality not only improves image database manipulation and management capability itself, but also enables faster and massive data processing inside the database.

PARALLEL PROCESSING

The other idea of implementing image processing inside the database is to leverage the Oracle parallel processing engine, which is the best of its kind in the IT industry.

Performance depends upon the design and implementation of the in-database processing strategy, the processing algorithms, speed of I/O, flexible memory utilization, to name a few. Given that modern computers are mostly multicore or have multiple CPUs, we think parallel processing should be implemented in any modern geospatial and image processing solutions.

Oracle database provides a powerful SQL parallel execution engine that can run almost any SQL-based operation – DDL, DML and queries – in the Oracle Database in parallel. When you execute a SQL statement in the Oracle Database it is decomposed into individual steps or row-sources, which are identified as separate lines in an execution plan (Dijcks, 2010). This is called parallel execution of SQL statements, which applies directly to all GeoRaster read-only functions such as metadata-related query operations and all single cell queries. However, with this parallel execution framework the individual raster processing functions, such as mosaic and raster algebra operations, cannot be directly parallelized without some special implementation. This is because each of the heavy image processing and raster manipulation operations is not purely row-based and has its own logic in how the raster data (or raster blocks) are internally processed.

We leverage the pipelined and parallel table functions to implement parallelism. The goal of a set of table functions is to build a parallel processing pipeline leveraging the parallel processing framework in the database (Oracle 2008. Dijcks, 2010). We encapsulate complex logic in a PL/SQL construct so that we can process different subsets of the data of a GeoRaster object in parallel. We begin with explicitly controlling the level of degree of parallelism (DOP) and deciding what subsets of the data to be handled in each subprocess. We used the output raster to split the whole region into subsets and the total number of subsets is decided by the DOP, which can be specified by users. Then our parallel execution framework will split the whole task into different subprocesses based on the total number of subsets and each subprocess will process one of the subsets independently. When all subsets are finished, the whole process is done.

We used this approach and have implemented parallel processing in all raster algebra functions, pyramiding and large-scale mosaicking. We conducted many tests on raster algebra and the performance improvement of parallelizing raster algebra functionalities are presented in another paper (Xie, 2012). We also conducted the following tests on the newly implemented parallelized large-scale mosaicking and pyramiding functions using the 24-CPU machine.

We used the same set of 31 Landsat 5 TM images. The total size of the source images is 11.9 GB. We first ran the GeoRaster mosaicking function mosaicSubset with different DOP and using different resampling. The resulting mosaic is 7.7GB in size. Then we ran the GeoRaster pyramiding function generatePyramid with different DOP on the 7.7GB mosaic image using different resampling approaches. The scripts are as follows:

```
-- mosaicking script:
-- select all 31 images that are stored in the tm_images georaster table
stmt := 'select grobj from tm_images where id > 0 order by id';
open cur for stmt;
-- mostly use default parameters to mosaick. this ensures the same functionality as gdalwarp is called
sdo_geor_aggr.mosaicsubset(cur, null, 32610, null, null, null, null, null, null, null,null,
   'nodata=true resampling=cubic', 'blocksize=(512, 512, 7)', gr, null, 'parallel=8');

-- pyramiding script:
 sdo_geor.generatepyramid(gr, 'resampling=cubic', null, 'parallel=8');
```

For mosaicking, the performance results are shown in Table 2. Figure 2 shows the speed changes when DOP increases. For pyramiding, the performance results are shown in Table 3. Figure 3 shows the speed changes when DOP increases.

As we stated, the source images overlap each other and almost half of the 31 images are in different projections. That means the mosaicking process involves image reprojection and resampling of the pixels. Pyramiding is mostly about resampling. So, both mosaicking and pyramiding are modestly complex processes.

To mosaick the 31 TM images into 1 image, it takes 18.65 minutes without parallelism but only 3.73 minutes with DOP = 8. Comparing with non-parallel processing, the average performance improvement is 4.86 times faster with DOP = 8. With DOP of 16 and 32, the average performance improvements are 3.21 and 3.91 times respectively. These are significant too but not as good as the improvement with DOP = 8. This is because mosaicking is too I/O intensive resulting in too much disk contentions among different subprocesses. The test machine has in total only 8 disks. Given better disk storage, we expect much faster performance. Another conclusion is that the overall performance improvement of mosaicking with cubic resampling is better than simpler NN, Bilinear and Biquadratic resampling. In most cases, the more complex the image processing algorithms, the better the performance benefit we can get from parallelism.

Table 2. Parallelized Mosaicking Execution Time in Minutes

Degree of Parallelism	**1**	**4**	**8**	**16**	**32**
MosaicSubset with NN	18.65	6.2	3.73	5.42	7.6
MosaicSubset with Bilinear	23.43	9.78	4.82	6.6	9.02
MosaicSubset with Biquadratic	32.55	13.87	7.05	7.6	9.52
MosaicSubset with Cubic	44.2	14.65	8.88	10.15	10.08

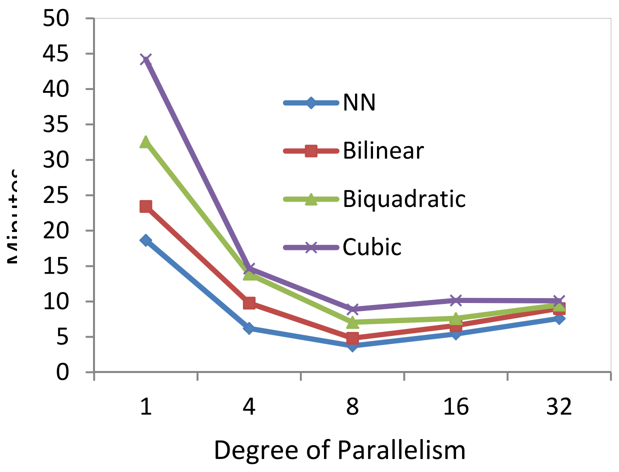

Figure 2. Parallelized Mosaicking with Different DOP

Table 3. Parallelized Pyramiding Execution Time in Minutes

Degree of Parallelism	**1**	**4**	**8**	**16**	**32**
generatePyramid with NN	2.67	1.07	0.5	0.65	0.52
generatePyramid with Bilinear	3.7	1.32	0.62	0.52	0.47
generatePyramid with Biquadratic	5.17	1.32	0.78	0.55	0.78
generatePyramid with Cubic	4.83	1.28	0.97	1.15	0.62

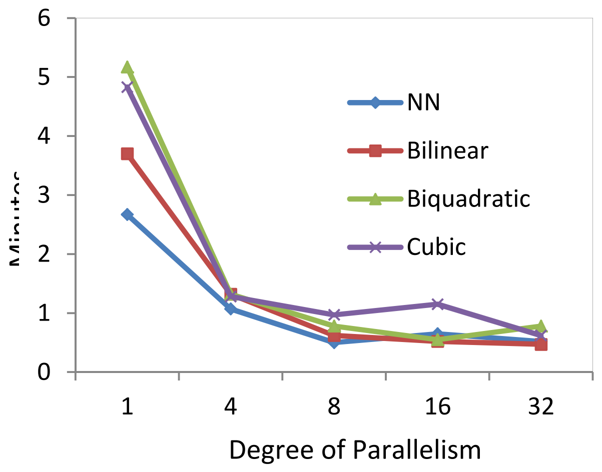

Figure 3. Parallelized Pyramiding with Different DOP

To pyramid the 7.7GB mosaic image, the average improvement is dramatic, ranging from 3.25 times faster with DOP = 4 to 6.86 times faster with DOP = 32 comparing with non-parallel processing. The pyramiding with biquadratic resampling with DOP = 16 is close to 10 times faster than that without parallelism. In this case, it takes only 0.55 minutes to generate the full pyramid for the single 7.7GB mosaic image.

In summary, our implementation of parallelism dramatically improves image processing performance inside the database. Parallelism is a key feature of our GeoRaster in-database image processing engine.

CONCURRENT PROCESSING

Another idea of implementing image processing inside the database is to leverage Oracle concurrent processing. Since we implement image processing inside the database and we fine-tune memory usage through our GeoRaster memory management infrastructure, Oracle concurrent processing capability becomes an immediate benefit. Concurrent processing is available to all GeoRaster functions directly, regardless the database is on a single machine or on a computer cluster.

We did a lot of tests on hundreds of concurrent image data queries, which are described in the paper (Xie, 2006). Those tests are more about database query. Unlike simpler image data queries, image processing involves more computation and more expensive disk read and write of large volume of image data. We want to make sure concurrency works well with them. So, this time, we conducted some tests on the concurrent image processing capability.

To test on a single machine, we used the same 24-CPU machine described in this paper. We chose the new GeoRaster mosaicking function mosaicSubset and 4 of the 31 Landsat images. The total source image size is about 1.52GB and the resulting mosaic size is 1.2GB (13240 x 13800 x 7 pixels). The resulting mosaic is shown in Figure 4. The 4 images are in different projections. The two images on the right hand side are in UTM zone 11N projection (SRID 32611) and the other two are in UTM zone 10N projection (SRID 32610). They also overlap. So, the mosaicking process involves complex computations.

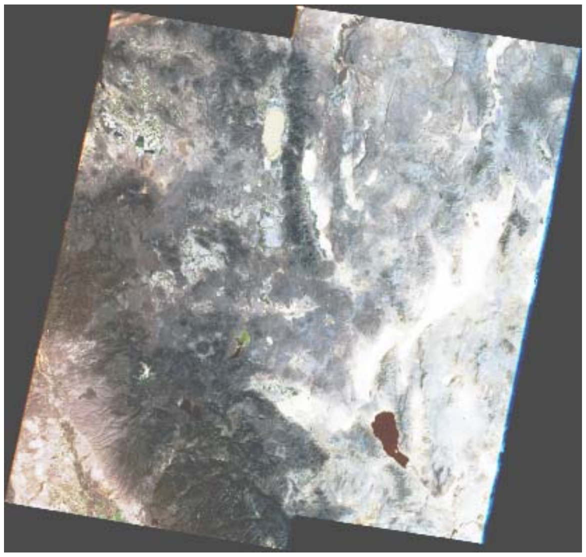

Figure 4. Mosaic of 4 Landsat Images.
(Image Courtesy of the U.S. Geological Survey)

We run the same mosaicking task to simulate concurrent tasks. The testing results are shown in Table 4 and Figure 5. In this concurrent processing test, the number of mosaicking tasks is the same as the number of concurrency. That means all tasks run concurrently in these tests.

As we can see, it only takes less than 4 minutes to finish 16 mosaicking tasks if concurrent processing is applied, while it takes about 41 minutes if sequential mosaicking is used. From Figure 5, when we run up to 16 concurrent mosaicking tasks the tasks don't really affect each other's performance much on this single machine. That means the tasks are fully leveraging the multi-CPU environment thus drastically improve throughput. Comparing with sequential processing, concurrent processing can finish the same tasks up to 11.3 times faster on the same machine.

Table 4. Concurrent and Sequential Mosaicking Execution Time in Minutes

Number of Mosaicking Tasks	**1**	**2**	**4**	**8**	**16**	**32**
Concurrent Processing	2.58	2.57	2.66	2.74	3.65	11.53
Sequential Processing	2.58	5.08	9.95	20.05	41.2	83.98

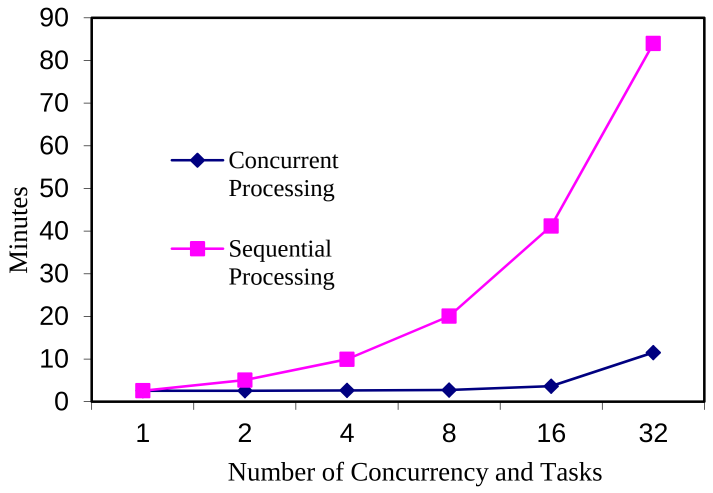

Figure 5. Sequential vs Concurrent Mosaicking.

The above test is on a single node computer. Oracle database Real Application Cluster (RAC) can distribute concurrent operations across multiple nodes in a cluster of machines. Instead of operating concurrent processing on a single node while other nodes are idle, the Oracle database can be configured to automatically spread concurrent processing across all available nodes, fully utilizing hardware resources to improve performance drastically. This is also called Oracle Enterprise GRID Computing, which allows you quickly process and analyze thousands of images and rasters stored in the GeoRaster database concurrently and on a global basis (Xie, 2006).

On a high-end computer, an Oracle Exadata Database Machine X2-2 Quarter Rack with Oracle Database 11gR2 software that has two RAC nodes, some initial experiments with our Oracle China consulting team show that (1) with a single process, the speed of exporting GeoRaster images using GDAL is 2.4 GB per minute; (2) with 200 concurrent threads on the two RAC nodes of the machine, the speed of direct exporting GeoRaster images using GDAL is about 2 TB per hour; (3) with 200 concurrent threads on the two RAC nodes of the machine, the speed of subsetting and clipping along irregular political boundaries (using GeoRaster's subset function) and then exporting the subsets using GDAL is about 1.1 TB per hour.

In summary, with both single machine and computer clusters, concurrent processing drastically improves massive image processing performance, database scalability and overall throughput.

CONCLUSIONS

Unprecedented data volume of geospatial imagery plus real time or near-real time processing requirements of such imagery dictate extreme scalability and performance of geospatial image database systems and processing solutions. Oracle Spatial GeoRaster takes an enterprise database-centric approach by enhancing Oracle database server to solve the database management challenges and achieve virtually unlimited scalability and great performance. The in-database image processing engine proposed in this paper enhances Oracle Spatial GeoRaster database management system by embedding more advanced image processing inside the database allowing data to be processed where the data is stored. It proves that complicated image processing can be successfully implemented inside the RDBMS databases. This in-database image processing approach avoids unnecessary data movement, enables parallel processing implementation and takes advantages of Oracle concurrent processing capabilities. The result is greater performance, better scalability and true security for all database management and image processing operations. The implemented image processing functions not only enhance database manipulation and query capabilities but also enable massive basic image processing directly inside the database. This removes the need of separate solutions for some remote sensing, GIS and business applications. For traditional remote sensing and GIS applications, specialized image processing packages and GIS solutions are still required. However, such third party solutions can also benefit greatly in performance from this image processing engine by pushing some basic data processing and filtering operations into the database so that less data is retrieved and transported into the client for further processing and analysis.

REFERENCES

Baumann, P., 2001. Web-enabled Raster GIS Services for Large Image and Map Databases. In: *5th Int'l Workshop on Query Processing and Multimedia Issues in Distributed Systems (QPMIDS'2001)*, Munich, Germany, September 3-4, 2001

Berger C., 2009. Oracle Data Mining 11g: Competing on In-Database Analytics, An Oracle Technical White Paper. http://www.oracle.com/technetwork/database/options/odm/oracledatamining11gwpv3-130702.pdf (16 March 2012)

Das J., 2010. Adding Competitive Muscle with In-Database Analytics. Database Trends and Applications. http://www.dbta.com/Articles/Editorial/Trends-and-applications/Adding-Competitive-Muscle-with-In-Database-Analytics-67126.aspx (16 March 2012)

Dijcks, J., H. Baer, and M. Colgan, 2010. Oracle Database Parallel Execution Fundamentals, An Oracle Technical Whitepaper. http://www.oracle.com/technetwork/articles/datawarehouse/twp-parallel-execution-fundamentals-133639.pdf (16 March 2012)

ESRI, 2005. Raster Data in ArcSDE® 9.1 - An ESRI White Paper. http://www.esri.com/library/whitepapers/pdfs/arcsde91-raster.pdf (16 March 2012)

GDAL, 2012. Image Reprojection and Warping Utility (http://www.gdal.org/gdalwarp.html)

Grimes, S., 2008. In-Database Analytics: A Passing Lane for Complex Analysis. Information Week. http://www.informationweek.com/news/software/bi/212500351?cid=RSSfeed_IE_News (16 March 2012)

Oracle, 2004. *Oracle Spatial GeoRaster, 10g Release 1 (10.1).*

Oracle, 2008. *Oracle Database Data Cartridge Developer's Guide 11g Release 1 (11.1).*

PostGIS, 2012. *PostGIS 2.0.2 Manual* (http://postgis.refractions.net/docs/index.html)

Xie, Q., Z. Li, and W. Xu, 2006. Using Enterprise Grid Computing Technologies to Manage Large-Scale Geoimage And Raster Databases. In: *the Proceedings of ASPRS 2006 Annual Conference*, Reno, Nevada, May 1 – 5, 2006.

Xie, Q., S. Ravada, W. Xu, and Z. Zhang, 2008a. An Enterprise Database-centric Approach for Geospatial Image Management and Processing. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXVII-B4.2008,* XXI ISPRS Congress, Beijing, China,

Xie, Q., 2008b. Oracle Spatial, Raster Data. *Encyclopedia of GIS*, Shashi Shekhar and Hui Xiong (editors), Springer. pp. 826 - 832.

Xie, Q., Zhang, Z., and S., Ravada, 2012. In-Database Raster Analytics: Map Algebra And Parallel Processing In Oracle Spatial Georaster. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XXXIX-B4, 2012*, XXII ISPRS Congress, Melbourne, Australia.