IBM Informix Dynamic Server Storage Optimization

Scott Lashley,
IDS Kernel/Performance Architect
(slashley@us.ibm.com)

IBM Informix Dynamic Server Storage Optimization
April 22, 2009

Note: Before using this information and the product it supports, read the information in "Notices".

Introduction

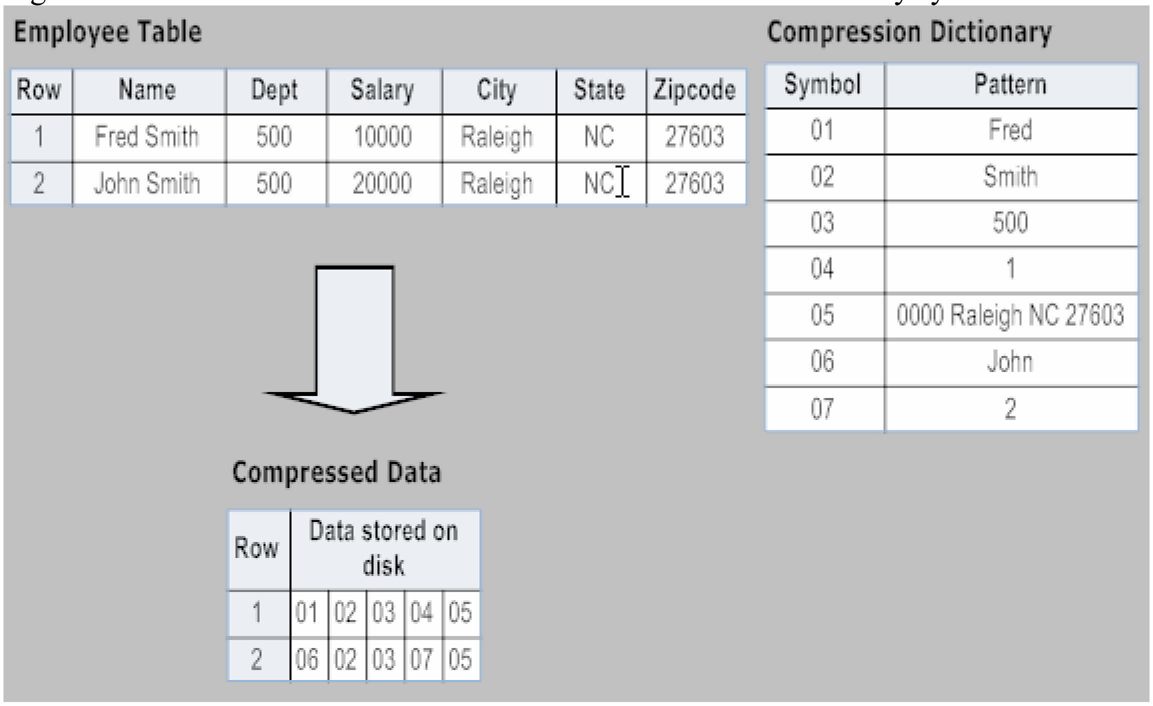
The amount of data stored within databases continues to grow rapidly. While the cost of storage continues to go down, it is not keeping pace with the amount of data generated. This puts pressure on IT budgets to find ways to reduce costs. When you consider that most databases employ redundant storage and backup copies of data, it is easy to see that even small database systems that use only a terabyte of storage can easily require 3 to 6 times that amount of total storage.

IBM® Informix® Dynamic Server (IDS) now offers data compression technology in the Storage Optimization Feature, available in April 2009 with the 11.50.xc4 release. IDS is sharing the same great technology first introduced in DB2® LUW which has been a huge success for DB2 customers. IDS provides full online support for turning on storage optimization and compressing existing table data while applications continue to use the table. This means that no system downtime is required to utilize the IDS storage optimization technology. Customers have been able to achieve up to 80 percent savings in storage, depending on their data characteristics. A reduction in data volumes also means less time to complete backup and restore operations. Many customers have experienced up to a 20 percent performance improvement in their applications due to less I/O and improved bufferpool utilization.

How does IDS compression technology work?

IDS compression technology works by considering the entire row and all its columns (except data such as BLOB objects that is stored outside the row as a single string of bytes. IDS identifies repeating patterns and stores those patterns as symbols in the dictionary. Once this dictionary is created, it is stored in a dictionary repository.

Figure 1 illustrates how row data in a table is converted into dictionary symbols.



Employee Table

Row	Name	Dept	Salary	City	State	Zipcode
1	Fred Smith	500	10000	Raleigh	NC	27603
2	John Smith	500	20000	Raleigh	NC	27603

Compression Dictionary

Symbol	Pattern
01	Fred
02	Smith
03	500
04	1
05	0000 Raleigh NC 27603
06	John
07	2

Compressed Data

Row	Data stored on disk				
1	01	02	03	04	05
2	06	02	03	07	05

Figure 1

The IDS storage optimization of data in a table or fragment consists of four steps:
1. Create the compression dictionary.
2. Compress the rows.
3. Coalesce the rows.
4. Reclaim free space.

Creating a compression dictionary

Dictionary creation involves sampling a set of rows from an existing table or table fragment and creating a dictionary of symbols that represent byte patterns. The dictionary is stored in the dictionary repository, which is included in the dbspace within the compressed partition. The dictionary also has an in-memory representation, so that active queries and updates can quickly compress and uncompress data, with minimal impact to performance.

This size of each dictionary is approximately 75 K bytes, but each dictionary can grow to be as large as 150 K bytes. Since the dictionaries are stored in memory (one dictionary per compressed partition), you must take into account these resource requirements.

Compressing the rows

After a dictionary is created for a table or table fragment, the process of compressing rows begins. Any new rows that are inserted or updated will be compressed automatically.

IDS compresses data as a background task, compressing each row and leaving the compressed data in the page that contains the row. Since this operation is executed in parallel with normal business transactions and queries, the compression work is managed in small transactions. The rows being actively compressed are only locked for a short duration, for minimal impact to normal business operations.

Coalescing the rows

Once a partition has been compressed, it is highly likely that there is a significant amount of unused space or 'holes' between the rows. The coalesce operation, also known as a repack operation, moves all of the rows to the front of the partition using a similar algorithm as the compress activity. It also utilizes small transactions and locks only those rows actively being moved.

Reclaiming free space

Lastly, once all the rows have been repacked, the shrink operation truncates off the unused portion of the partition, and returns the space back to the dbspace where the partition is located.

Modular Design

Each of the four operations listed above can be activated in isolation. This means that regular, uncompressed tables can be repacked, or shrunk, or both. A compressed table can be repacked at any time, any number of times over the course of its life, all without preventing other update activity. This modular, low-impact design, combined with a SQL interface, provides administrators maximum flexibility and advantage, and sets the stage for many useful autonomic features in the future.

Utilizing IDS Storage Optimization via OAT

OAT (Online Administration Tool) provides the DBA with an easy interface to identify compression candidates. The DBA can use OAT to view all the tables within a selected database. After the database is selected, OAT selects the tables within that database that might benefit from being compressed and calculates the approximate space savings. If you hold the cursor over the ***Usage*** column for a given table, OAT displays a compression estimate.

Figure 2 illustrates the OAT database view with a compression estimate.

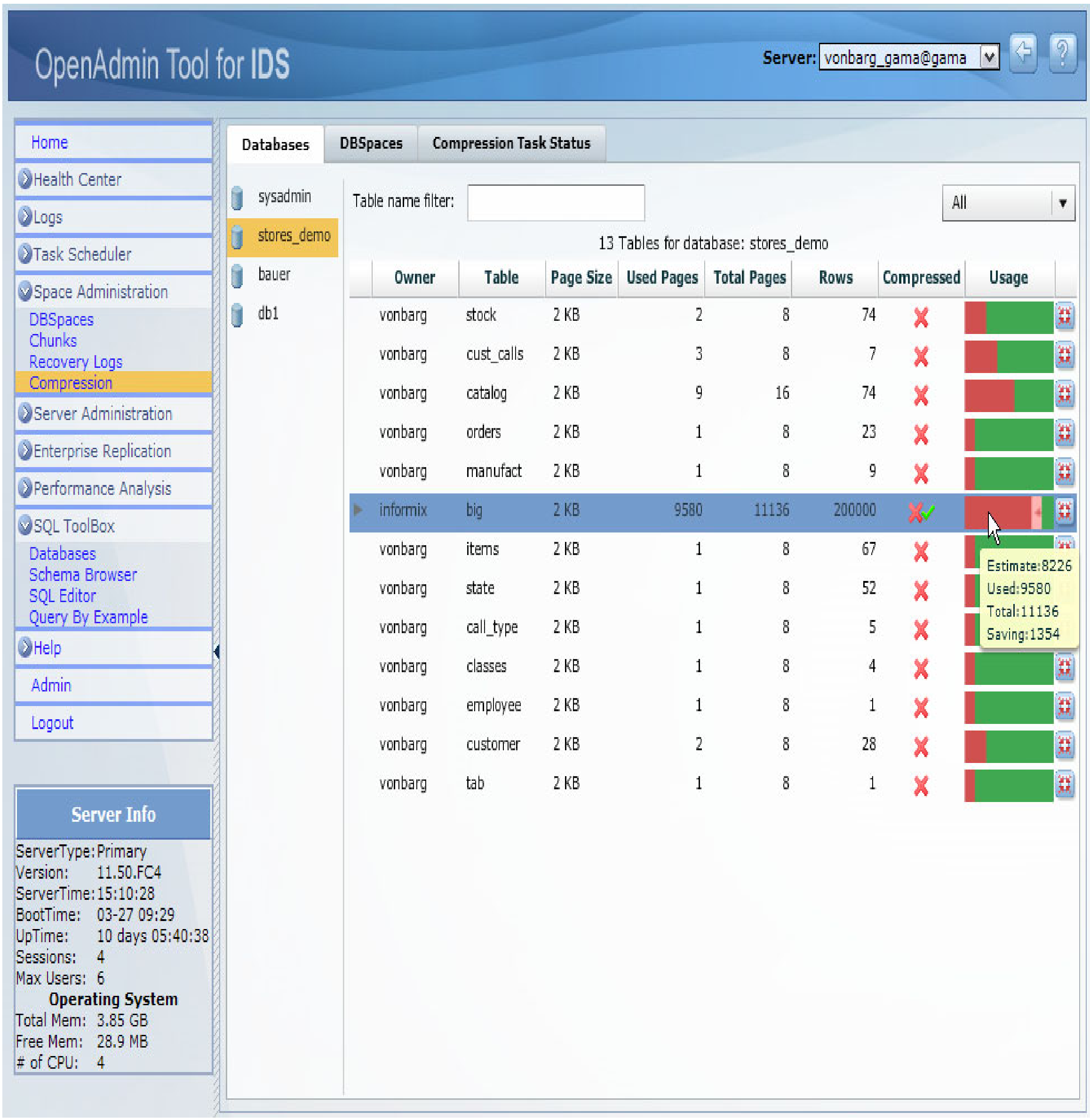


Figure 2

Figure 3 illustrates the dbspace view of compression. Here, all the fragments of tables that have a partitions located in the chosen dbspace are displayed.

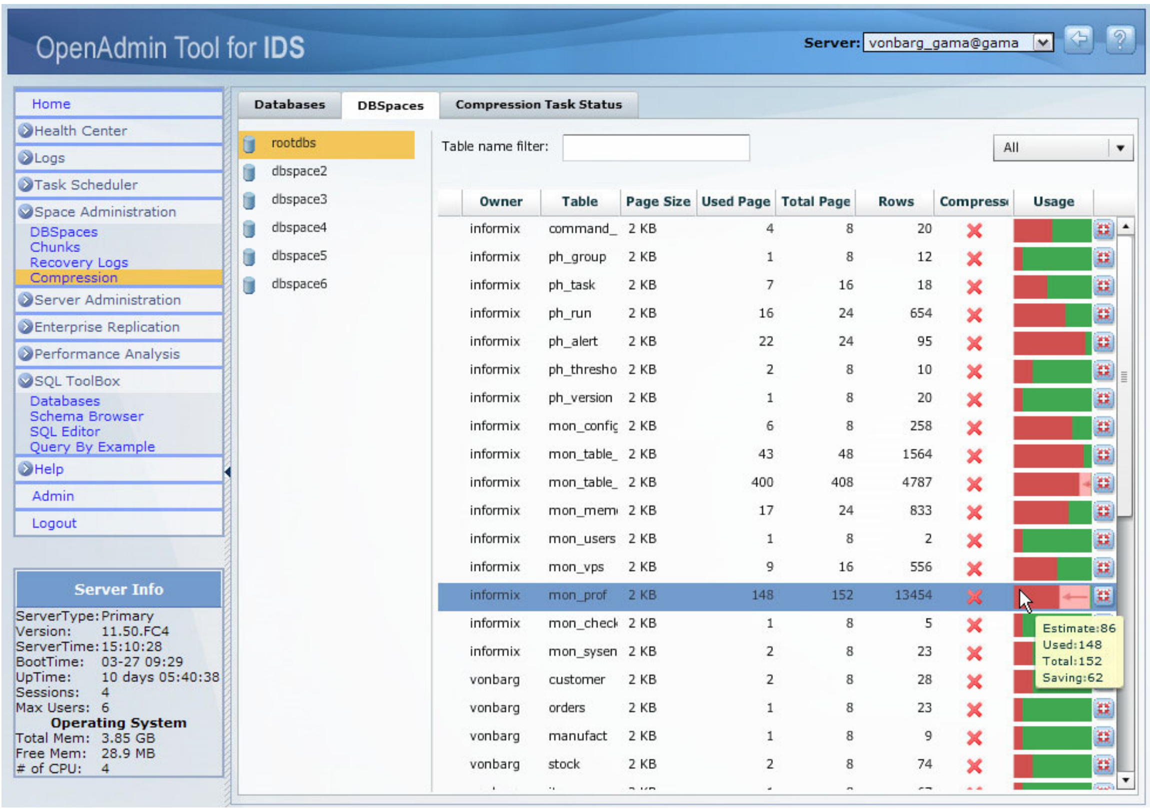


Figure 3

Figure 4 illustrates how you can choose to compress only part of a table. A fragmented table is indicated by the ► symbol in front of the owner of the table. If you click that symbol, each of the fragments of the table will be displayed. You can then choose which fragment to compress.

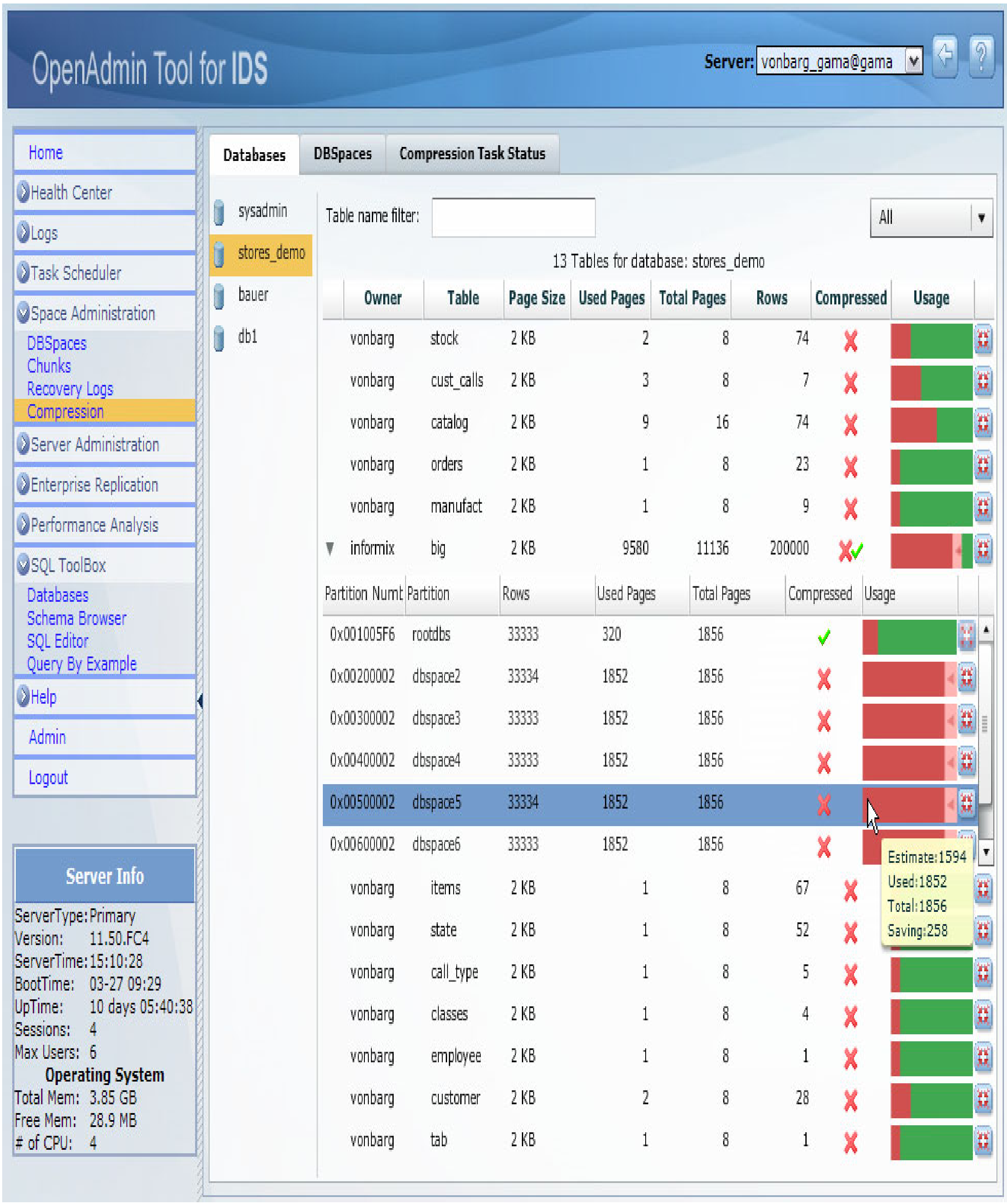


Figure 4

After you choose to compress a table or fragment, a new screen appears. You can use this screen, as illustrated in Figure 5, to choose the actions to execute on a specific fragment or the entire table.

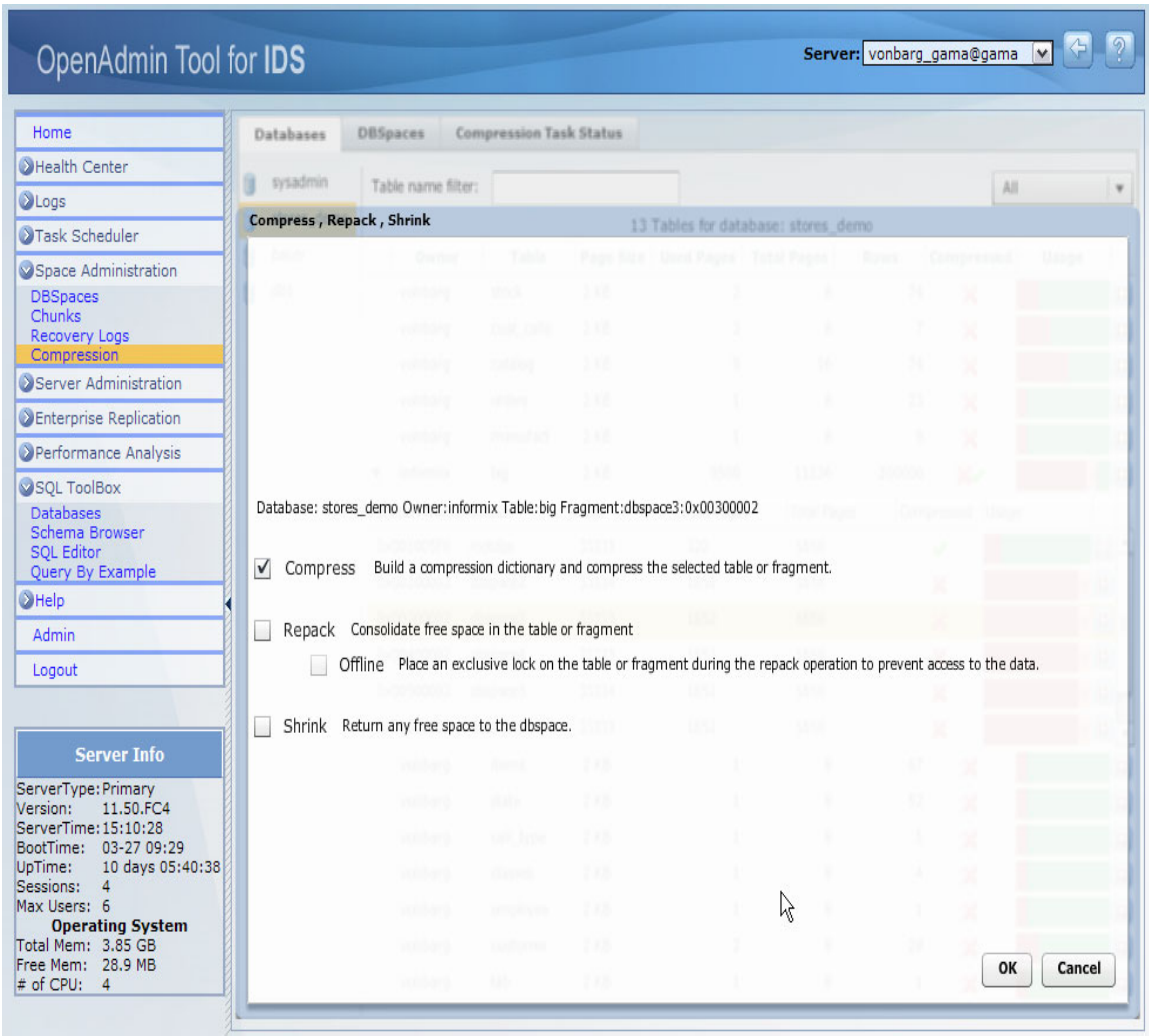


Figure 5

After you choose which operations to perform on a table or fragment, IDS runs the job. You can immediately go to the task list to see the status of running tasks. Figure 6 shows the status of a compression job that is running on a table.

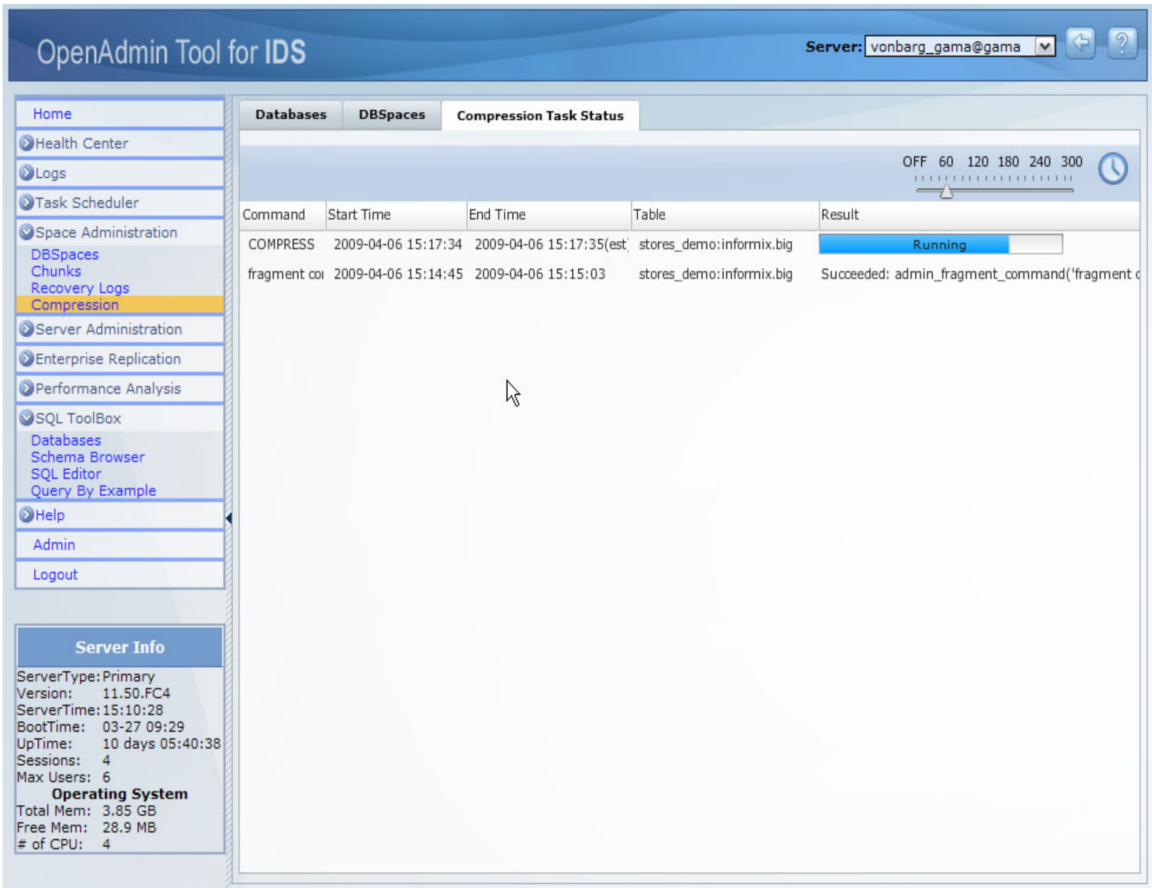


Figure 6

Automatic Compress

IDS also provides a way to allow the server to automatically find good compression candidates and compress them. OAT allows you to configure a task to periodically search all the partitions contained within the IDS instance. If this task finds fragments that qualify as good compression candidates, the task automatically submits a job to compress, repack and shrink the fragments. Before a job is created and launched, you can set job qualifications based on a minimum size and a minimum compression ratio. Figure 7 illustrates the how to enable an automatic compression task.

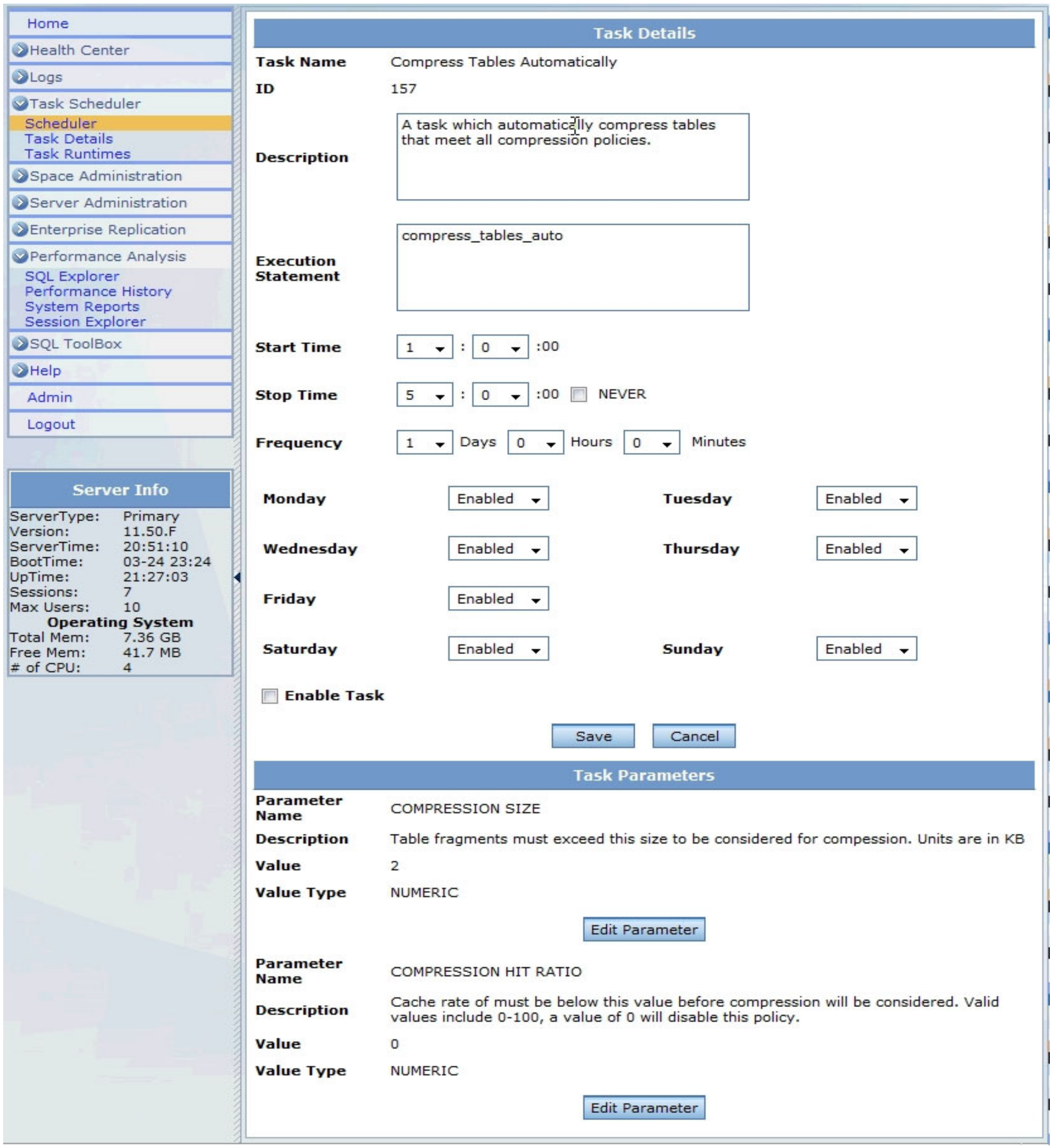


Figure 7

Figure 8 illustrates what a compression task looks like when viewing the task scheduler in OAT.

Figure 8

When to use data IDS Storage Optimization?

IDS Storage Optimization can help to significantly reduce storage costs.
A typical database can require quite a bit more storage than one might perceive. Consider the following scenario:

> Customer X has a 2 Tb system. Within that system, 1 Tb is a candidate for compression and it has a compression ratio of 50 percent. The initial storage savings for that system starts at 500 Gb. The system also generates 20 Gb of log data each day, and the customer keeps 30 days worth of logs. Since IDS compresses the log data for compressed tables, let's assume that 10 percent of that log storage can be saved. This results in savings of 2 Gb per day, or around 60 Gb per month. If Customer X needs 3 complete backups of the data, another 1.5 Tb can be saved, since the compressed data takes less backup space.
>
> If you add it all up:
>
> 500 Gb less storage for the database
> 60 Gb less storage for log backups
> 1500 Gb less storage for archives
> ===============================
> 2060 Gb total storage savings

This is the immediate storage space saved! When you consider ongoing data growth with the total cost of storage, these savings can add up even more.

IDS Storage Optimization can improve performance of data scans

Data compression ratios

The IDS Storage Optimization feature is very effective in reducing the storage requirements for tables. Figure 9 below shows data from various sources, and how effectively the IDS Storage Optimization feature reduced storage requirements.

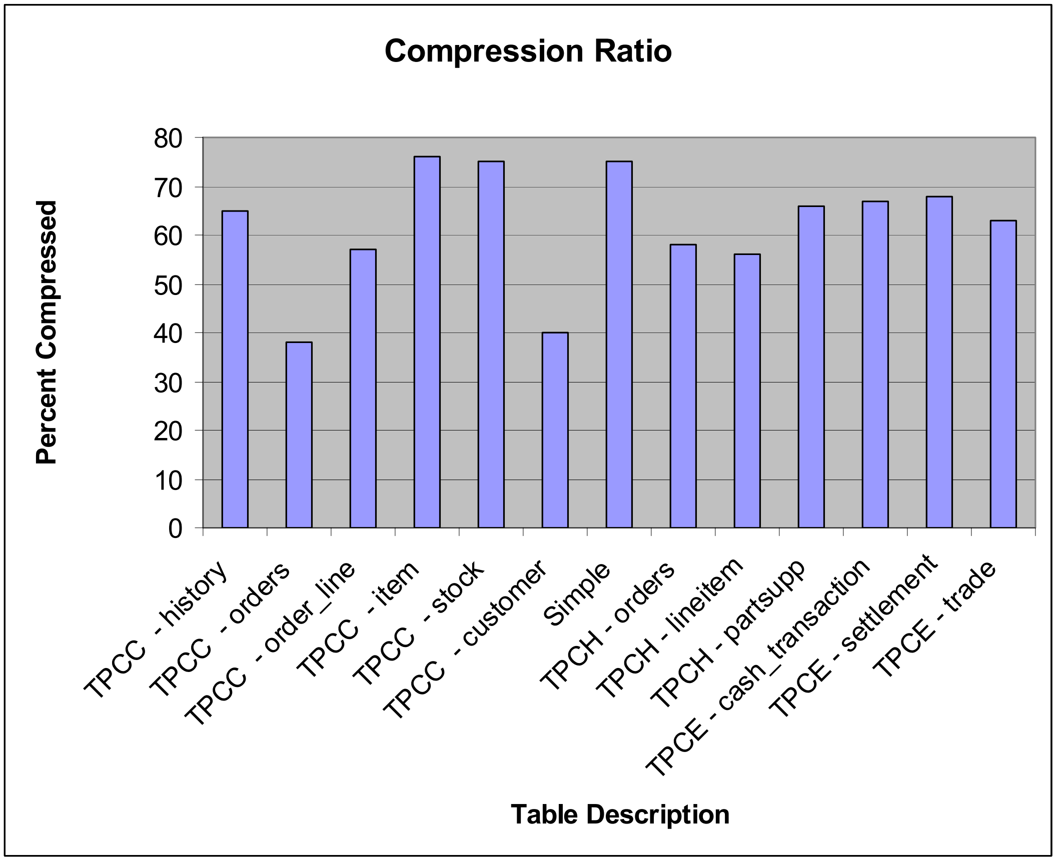


Figure 9

As this figure suggests, IDS Storage Optimization was able to reduce the size of a table from 40 to 80%.

In recent validation testing, one financial services customer reported a very successful reduction of an 8 Tb database to just under 2 Tb. The customers were thrilled because not only did they save storage, but their backups were significantly faster because of the reduction in the number of I/Os since the database size had been reduced significantly.

To see how your data might compress, you can find a Storage Compression Estimation tool at ibm.com/informix/compression. This tool takes information from your existing infrastructure and can estimate the compression ratio for your data so you can estimate the savings in your environment.

Data scans across compressed data

Compressed data requires less storage, which also translates into less I/O for data scans. The performance gain is obvious, given that data scans will now require less physical I/O to complete the scan. The amount of performance improvement will vary, depending how the scanned data is processed. In cases where the processing of the data is CPU-bound, doing less I/O will not improve the performance.

Database backup performance will improve

Database backups can also experience significant performance improvements, because the size of the database is significantly smaller. This means few I/O operations leading to faster backups. It is important to note that for large tables, the repack and shrink options must be executed to coalesce the data. This removes unused extents from the end of the table.

Improving cache hit ratio

Each application has something called the *working set*. The *working set* is the set of data that the application requires across a particular interval of time. The more of the working set that is contained in memory, the better the application will perform, because it has to perform less I/O. When the *working set* fits into the bufferpool, IDS Storage Optimization will not make a difference (and could even impact performance). But when the *working set* does not fit into the bufferpool, compressing the data can significantly improve performance, because more of the *working set* fits into the bufferpool and less I/O is required

In the example shown in Figure 10 below, the TPCC benchmark application demonstrates improved performance with compressed data. In the TPCC benchmark all data is equal, and therefore all of the data is part of the working set. The growth of the working set below is simulated by increasing the number of warehouses and the number of users. This increases the size of all of the tables and grows the working set. This is similar to a business that is growing with more application sessions and more data to manage. In this experiment, the amount of memory and CPU processing power is kept constant. As the working set grows, the number of I/Os increase, which impacts overall throughput.

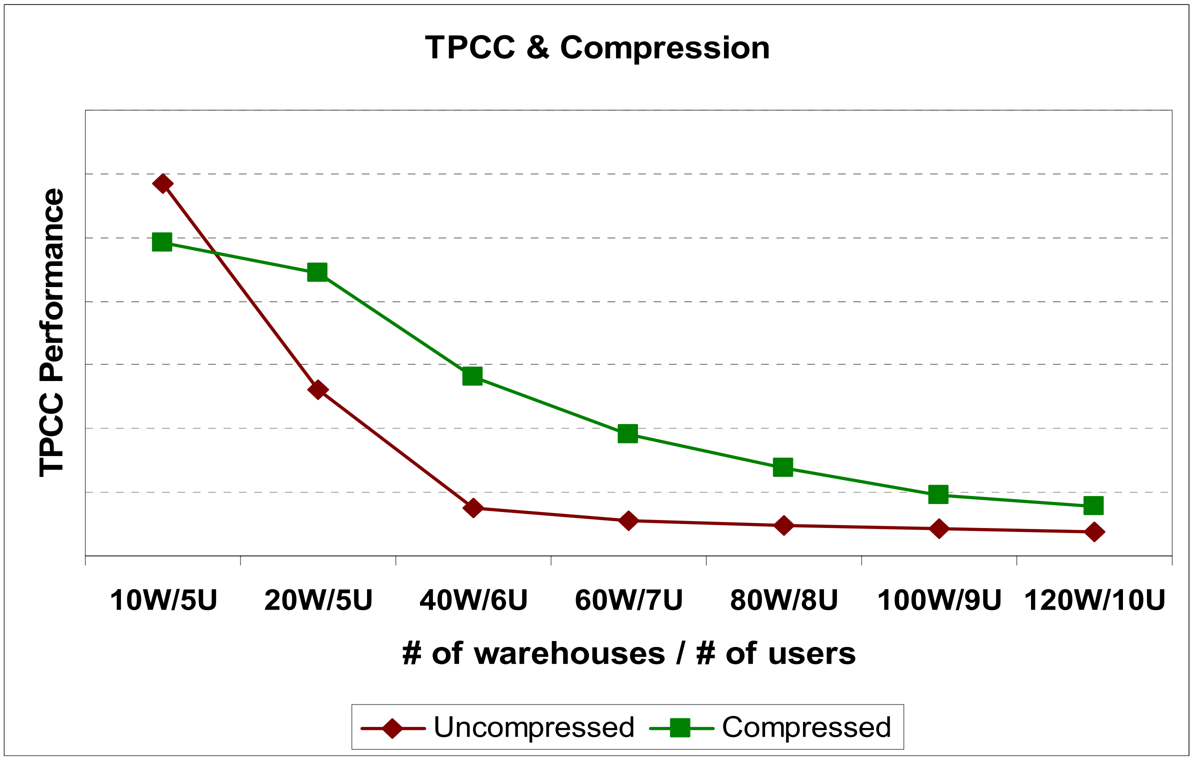


Figure 10

The example in Figure 10 shows a workload that started with 10 warehouses and 5 users. Because the workload is totally cached, and does no I/O other than transaction logging, the overall throughput is significantly better using uncompressed storage. This is because of the cost to materialize the row with each row access. But, as I/O is introduced into the workload, compressing data can improve performance. More of the working set fits into the buffer cache and less I/O occurs. As the processes become more and more I/O bound, the difference between compressed and uncompressed transaction throughput begins to taper off.

It should be noted, none of the experiments were done with the system being fully CPU bound in the cached case or fully I/O- bound in the non cached cases.

Another way that compression can help improve performance is by helping applications maintain good response times. Often, applications must adhere to strict service level agreements (SLAs), where the application must sustain a nominal response time for every transaction. As a database grows and the working set cannot be maintained within the buffer cache, there can be a dramatic effect on response time because the speed of I/O is considerably slower than a memory access.

Figure 11 below shows how compression can help maintain response times as the size of the working set grows, making it easier for applications to meet their performance service level agreements.

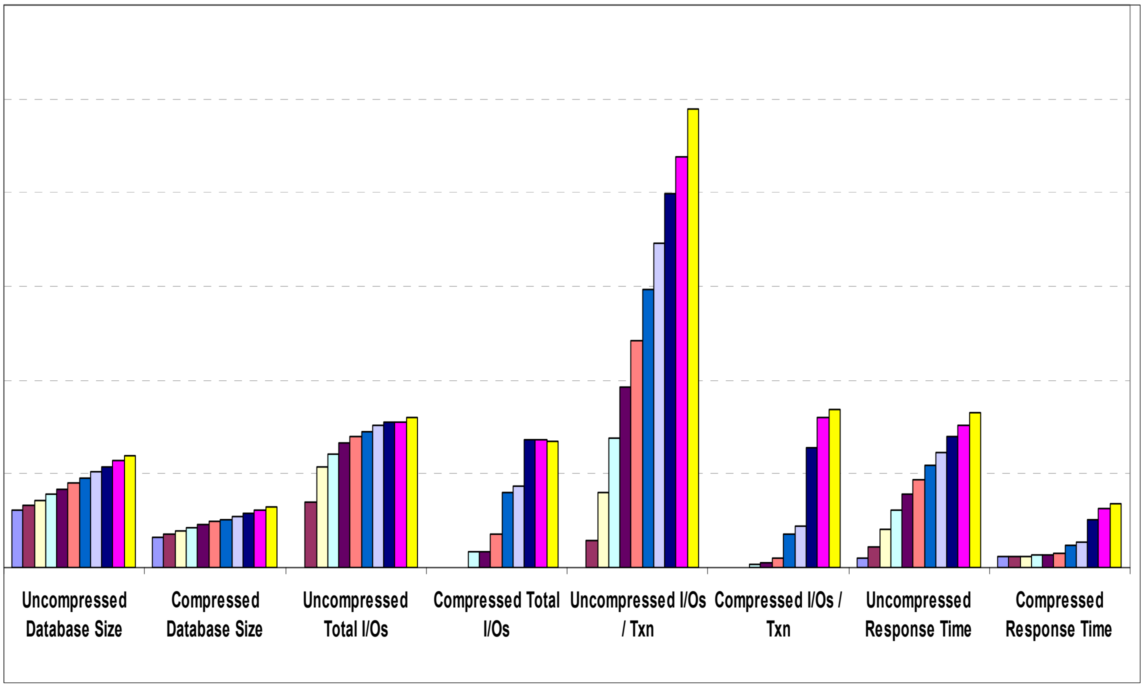


Figure 11

Figure 11 shows how the increasing I/O that is triggered by database growth causes transaction response time to grow rapidly. Implementing compression can help slow down the response degradation. Each of the bars in Figure 11 represents an experiment in which the database size of the compressed data is about ½ the size of the uncompressed database. With each experiment, the database size was increased. As the database size increased, so did the overall I/Os in the system as well as the I/Os per transaction.

Large rows that are split across pages

When a row is too large to fit on a page, it requires multiple I/Os to retrieve all the pieces of the row. Compression can often be used to reduce the size of the row so that the entire row fits within a single page.

You can use the **oncheck –pT** command to display the number of remainder pages a table is using before and after IDS Storage Optimization is applied. The information displayed in the “Compressed Data Summary” section of the output shows the number of any compressed rows in a table or table fragment and the percentage of rows that are compressed. If rows are not compressed, the "Compressed Data Summary" section does not appear in the output.

Storage Optimization Considerations

Cached tables

There are some situations in which compression should be avoided. When a table is already cached, the use of Compression on that table can degrade performance. There is an added overhead to materialize the uncompressed version of the row with each row access. This adds an additional CPU cost.

Random access across a huge data set with no clustering

In some other situations, a performance improvement will not be visible even when there are significant storage savings. These situations can be evaluated to determine the overall payoff for implementing compression technology. For example, when performing OLTP style queries across a huge data set, where the data is not clustered, IDS Storage Optimization might not improve performance. Consider the following example:

- Table T1 has 100 M rows across 1 M pages (100 rows per page)
- The application randomly accesses 100 rows / sec from the table

Give the above criteria, there is one chance in a million that a given row operation will access a particular page. Since all accesses have an equal chance, it is unlikely that there will be enough accesses to the table that they will hit the same page while the page is in the buffer pool. IDS Storage Optimization is still a great mechanism to reduce storage costs, as noted above, but in this case, where the working set is huge, it simply won't help improve performance.

Summary

The IDS Storage Optimization feature will allow customers to experience significant storage savings. In keeping with the long-time tradition of making IDS technology easy to use, IBM has made the interface to this feature simple or even automatic through OAT. And because all the options can be executed online, there is very little impact to existing applications when this technology is deployed to production systems.

Edited by
Anup Nair, John Lengyel, Cathy Elliot, Patricia Smith

Acknowledgements
Thanks to Monish Gupta, Vladimir Kolobrodov, Jeffrey McMahon and Jun Shan for providing assistance in providing experimental data.

Notices