Beyond PostgreSQL – A Look Inside Postgres Plus Advanced Server

An EnterpriseDB White Paper

for DBAs, Application Developers, and Enterprise Architects

June, 2009

http://www.enterprisedb.com

EnterpriseDB™

Executive Summary

For more than 20 years, PostgreSQL has earned the reputation as the world's most advanced open source database. EnterpriseDB's Postgres Plus Advanced Server builds on the rock-solid PostgreSQL foundation to deliver significant value-added features not available in the community edition.

The capabilities added to PostgreSQL by EnterpriseDB address a wide range of enterprise requirements, including application development enhanced performance, runtime management, usability, and scalability. In addition, EnterpriseDB provides 24x7 "follow the sun" technical support, as well as training and professional services needed by application developers, DBAs and global operations teams.

Organizations seeking to leverage the exceptional price/performance of an enterprise-class open source database, including current users of PostgreSQL, are well served by Postgres Plus Advanced Server and the services designed to optimize and guarantee continued performance.

An in depth discussion targeted specifically to your organization's requirements can be scheduled with an EnterpriseDB domain expert by sending an email to sales@enterprisedb.com.

Overview

Postgres Plus Advanced Server is based upon and developed alongside the open source community edition of PostgreSQL. Over 20 years of development have designed it for today's most demanding database applications. Although Postgres is renowned for its transactional capabilities, it also performs exceptionally well across a spectrum of inquiry intensive and mixed load applications.

Interestingly, from a development point of view, Advanced Server is usually a version behind the community edition. EnterpriseDB uses this lag to harden and polish Advanced Server to meet the performance, stability and scalability needs of a broad range of enterprise applications.

While maintaining a buffer from the bleeding edge, EnterpriseDB is also an active member of the PostgreSQL community and contributes significantly to the evolution of key PostgreSQL technologies. The Company employs leading community members and sponsors major features such as Heap Only Tuples, Fast Text Search, StackBuilder, and even administrative and development tools like the multi-platform pgAdmin III. In addition, EnterpriseDB partners with other community leaders such as NTT (Nippon Telephone & Telegraph) to co-sponsor innovative PostgreSQL features such as Synchronous Replication. EnterpriseDB also independently manages the GridSQL open source project, which delivers high-powered parallel query capabilities to PostgreSQL users.

There are many features that distinguish Postgres Plus Advanced Server from PostgreSQL. One feature which will not be covered in this paper is Oracle compatibility. It is worth mentioning though because many of the key Oracle compatibility features in Postgres Plus Advanced Server are essential (and available) for many types of Postgres applications. These features include Query Optimizer Hints, EDB*Loader, and Bulk Data Collection.

The primary feature differences between PostgreSQL and Postgres Plus Advanced Server discussed in this paper can be summarized as follows:

- Performance
  - DynaTune™
  - Query Optimizer Hints
  - EDB*Loader
  - Bulk SQL Operations
  - Query Profiler
  - Asynchronous Pre-Fetch
- Management
  - DBA Management Server

    - DBA Monitoring Console
    - Dynamic Runtime Instrumentation Tools Architecture
    - Update Service
    - Audit Logging
    - Database Migration
- Usability
    - Enterprise Class Packaging
    - PL Debugger
    - Transaction Error Recovery
- Scalability
    - GridSQL
    - Infinite Cache
    - Database Links

By adding these features to PostgreSQL, EnterpriseDB delivers an enterprise class database platform that provides lightning fast performance, near-linear scalability, 99.99% availability, rock-solid security, support for complex transaction processing, and the ability to replicate database across the globe in near real time – all in a remarkably simple to install and easy to use package with incredibly low Total Cost of Ownership.

The remaining sections of this document provide an overview of Advanced Server's value-added features over and above the PostgreSQL foundation.

Performance

Postgres Plus Advanced Server's value-added performance enhancements are derived primarily from EnterpriseDB's extensive investments in Oracle compatibility. Many of the features originally created to support Oracle compatibility are used extensively today in non-Oracle applications, including Optimizer Hints, EDB*Loader and Bulk SQL Operations. For a detailed discussion of all the Oracle features available in Postgres Plus Advanced Server refer to the EnterpriseDB white paper titled: Delivering Oracle Compatibility.

DynaTune™

EnterpriseDB's DynaTune™ with Workload Profiling simplifies one of the most difficult tasks faced by DBAs and application developers: tuning your database properly as a function of its utilization of the Server hardware and the application's utilization of the database.

The more popular databases seem to subscribe to the tuning mantra of "more tuning options are better". Yet database experts often note that a multitude of tuning options frequently discourage DBAs from implementing proper optimizations. Thus, the best tuning configurations result from combining well known large grained tuning adjustments and then fine tuning from there, rather than trying to individually manage numerous tuning parameters, many of which are obscure to all but a small group of performance domain experts.

DynaTune removes the uncertainty from Postgres database tuning. DynaTune uses a sophisticated algorithm developed by extensive performance testing using a wide range of hardware options to set multiple Postgres configuration options based on user inputs for two fundamental usage questions. This predictable install time tuning assures users of a proper foundation for good performance, which can be further tuned as their application usage matures.

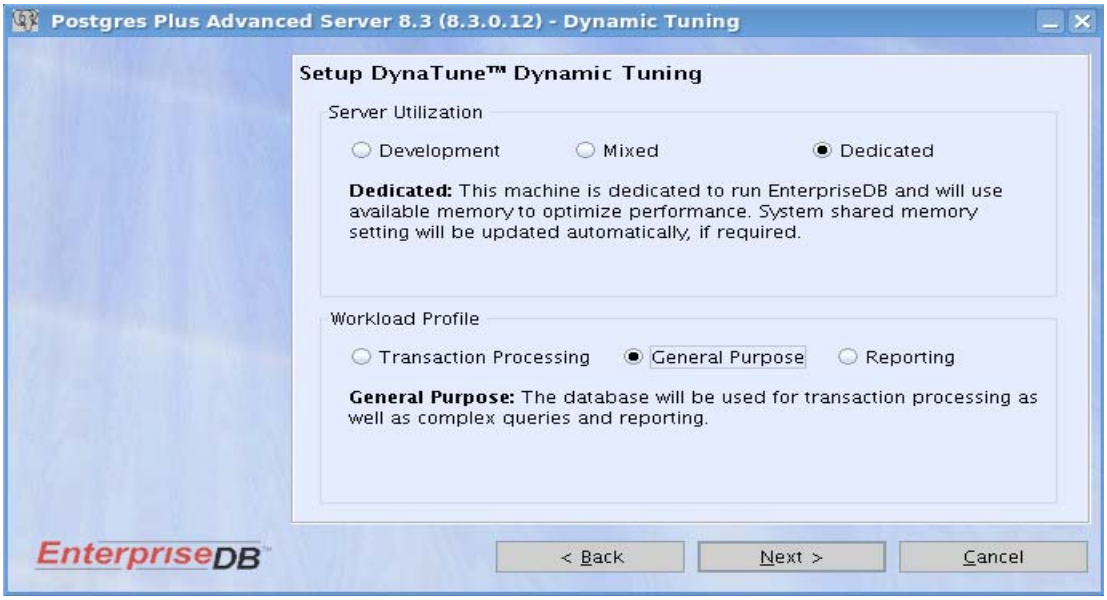


Figure 1: DynaTune setup in the installer

In addition, after hardware upgrades (e.g., memory expansion), DynaTune will automatically recalibrate tuning parameters on database startup to utilize the new configuration in a manner consistent with the previous settings.

Query Optimizer Hints

Query Optimizer Hints allow application developers to influence the chosen execution plan of the SQL optimizer. This is useful in cases where a human may know information about the data that the optimizer does not know. For example, a certain index may be more selective for a certain query.

A common scenario in a PostgreSQL world is the inconsistency of execution times of prepared queries. Since PostgreSQL will plan a prepared query the first time it is run, the plan may be sub-optimal in the majority of the subsequent execution cases. For example, query plans created on small tables often become sub-optimal as those tables grow over time. PostgreSQL favors a sequential scan to retrieve the data on small tables and index scans on large tables.

Consider the case of a common commercial database application – the daily transaction table. At the beginning of each day, there may only be a handful of rows in the table; by the end of the day, the table may contain millions of rows. A query plan prepared at the beginning of the day will quickly become sub-optimal. An application developer who understands this growth profile can override Postgres' default plan by applying a query optimizer hint to use an index scan on the daily transaction table.

An optimizer hint is easily added to the select clause of a query as shown in the example below.

```
PREPARE foo(int) AS
SELECT /*+ INDEX(emp emp_pk) */ empno, ename, dname
  FROM emp, dept
 WHERE emp.deptno = dept.deptno
   AND empno = $1

EXECUTE foo(7369);
```

The above query produces the query plan below. The "emp_pk" index is used to execute the query when a sequential scan of the "emp" table would normally be preferred.

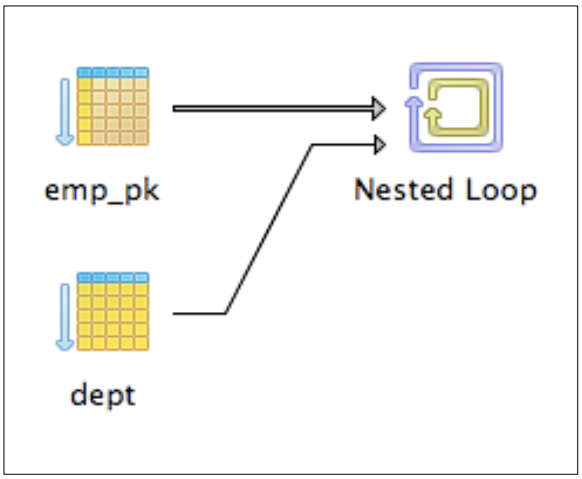


Figure 2: An optimized query plan

There are many types of query optimizer hints available to tailor the execution plan to the needs of the application. The query can be optimized for a plan that returns the initial rows quickly or to use a certain type of join. These hints are all intended to give application developers additional flexibility and control over query execution to assure consistent application performance and end user experience.

The table below enumerates the query optimizer hints available in Postgres Plus Advanced Server.

Query Hint	Description
ALL_ROWS	Optimizes for retrieval of all rows of the result set.
CHOOSE	Does no default optimization based on assumed number of rows to be retrieved from the result set. This is the default.
FIRST_ROWS	Optimizes for retrieval of only the first row of the result set.
FIRST_ROWS_10	Optimizes for retrieval of the first 10 rows of the results set.
FIRST_ROWS_100	Optimizes for retrieval of the first 100 rows of the result set.
FIRST_ROWS_1000	Optimizes for retrieval of the first 1000 rows of the result set.
FIRST_ROWS(n)	Optimizes for retrieval of the first n rows of the result set.
FULL(table)	Perform a full sequential scan on table.
INDEX(table [index] […])	Use index on table to access the relation.
NO_INDEX(table [index] […])	Do not use index on table to access the relation.
USE_HASH(table […])	Use a hash join created from the join attributes of table.
NO_USE_HASH(table […])	Do not use a hash join created from the join attributes of table.
USE_MERGE(table […])	Use a merge sort join for table.
NO_USE_MERGE(table […])	Do not use a merge sort join for table.

USE_NL(table […])	Use a nested loop join for table.
NO_USE_NL(table […])	Do not use a nested loop join for table.

Table 1: Query Optimizer Hints available in Postgres Plus Advanced Server

EDB*Loader

EDB*Loader has several features that improve the performance of loading basic text files into a Postgres database. The PostgreSQL standard for this functionality is the “COPY” command, but it has important limitations that can significantly increase loading times.

The first limitation is the inability of the “COPY” command to handle fixed width file formats. Many files are in this format but, using PostgreSQL, they must be translated into a delimited format causing the file to essentially be written twice.

The other major limitation of the “COPY” command is the lack of error handling. When using the “COPY” command, if a single row of the file is invalid all of the rows are rejected, requiring the whole file to be reprocessed.

EDB*Loader has the ability to move invalid records to a discard file, allowing file processing to continue when load errors occur. Users can separately inspect the discard file and resolve specific data load issues once the bulk load process has completed.

However, the feature in EDB*Loader that produces the most dramatic increase in load performance is the ability to perform a Direct Path load. A Direct Path load bypasses some of the processing overhead that slows down bulk load performance and also requires an all or nothing completion approach.

A Direct Path load parses the input rows according to the column specifications defined in a control file. It then converts the file data into the column data type and builds an internal data structure. The data structure is then converted directly into a Postgres data block format. The newly created blocks are written directly to the data directory, bypassing most processing and resulting in much faster load times.

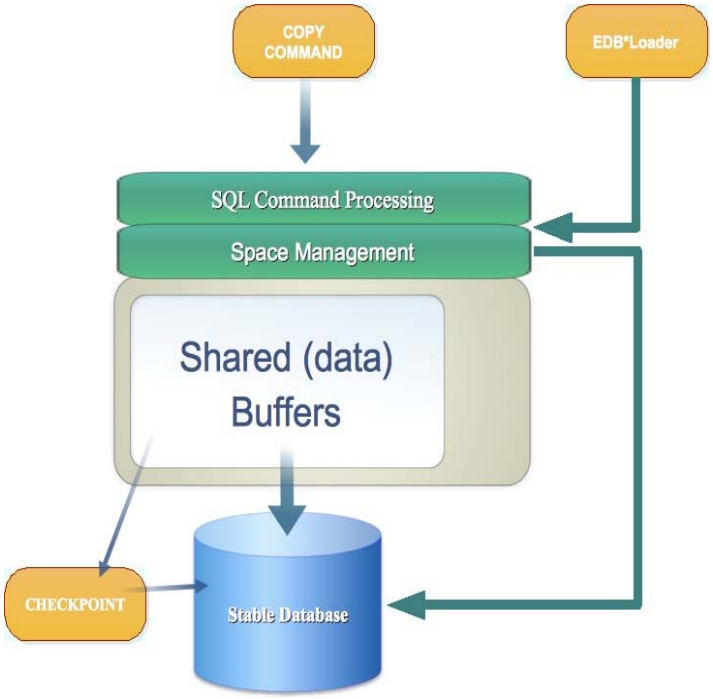


Figure 3: Copy vs. Direct Load of data

The differences between the Copy loading method and the Direct Path loading method are contrasted in Figure 3.

Bulk SQL Operations

SQL commands that return result sets consisting of large numbers of rows may not operate as efficiently as possible due to the constant context switching that must occur between the database server engine and the procedural language code in order to transfer massive result sets.

This inefficiency can be mitigated by using a collection to gather result sets into memory which the client can then access. Postgres Plus Advanced Server's stored procedure language (SPL) includes the BULK COLLECT clause which is used to specify the aggregation of large result sets into an in-memory collection. By changing PL/pgSQL functions that manipulate large cursors to SPL functions using BULK COLLECT, performance can be increased by 100% or more.

To demonstrate the use of BULK COLLECT, consider the function *samplefunction1()* in the code fragment below.

```
CREATE TABLE tab1 AS
SELECT * FROM accounts  WHERE 1=0;

CREATE OR REPLACE FUNCTION samplefunction1() RETURNS VOID
```

```
AS $$
DECLARE
  arow  accounts%ROWTYPE;
  acur  CURSOR FOR
          SELECT * FROM accounts;
BEGIN
  OPEN acur;
  LOOP
    FETCH acur INTO arow;
    EXIT WHEN NOT FOUND;
    INSERT INTO tab1 (aid, bid, abalance, filler)
    VALUES (arow.aid, arow.bid, arow.abalance, arow.filler);
  END LOOP;
  CLOSE acur;
END;
$$ LANGUAGE 'plpgsql';

SELECT samplefunction1();
```

Function *samplefunction1* demonstrates a simple cursor that manipulates a large result set. The function *samplefunction2* below accomplishes the same results with an 85% performance gain. BULK COLLECT allows developers to selectively target long running functions and dramatically improve performance.

```
CREATE OR REPLACE FUNCTION samplefunction2 RETURN VOID
AS
DECLARE
    TYPE atab_type IS TABLE OF accounts%ROWTYPE INDEX BY
BINARY_INTEGER;
    atab atab_type;
    CURSOR acur IS SELECT * FROM accounts;
BEGIN
    OPEN acur;
    LOOP
      FETCH acur BULK COLLECT INTO atab;
      FORALL i IN 1..atab.COUNT
      INSERT INTO tab1 VALUES (atab(i).aid, atab(i).bid,
atab(i).abalance, atab(i).filler);
      EXIT WHEN acur%NOTFOUND;
    END LOOP;
    CLOSE acur;
END;

SELECT samplefunction2();
```

Query Profiler

Postgres Plus Advanced Server contains a Query Profiler in the DBA Management Server console application. The Query Profiler examines a

selected database's activity and generates a report of the SQL commands executed against that database. A query Profiler report can help database administrators improve the performance of their systems. The information reported by Query Profiler can be utilized for two primary purposes:

- Tracking down the longest running SQL commands so they can be improved.
- Tracking down frequently used SQL commands or time consuming operations that can be considered for further performance optimization.

The resulting list of SQL commands can be ordered on the basis of Total Execution Time, Average Execution Time, or Statement Count (the number of times a given SQL command was repeated) by selecting the desired ranking from the Order By drop down list.

The types of SQL commands that appear can be filtered by checking the appropriate boxes:

- Include Inserts
- Include Updates
- Include Deletes
- Include Selects
- Include Others

Below is a sample of the output available from the Query Profiler in the DBA Management Server.

Results

Rank	Count	Total Execution Time(ms)	Average Execution Time(ms)	Query
1	121	30,431.004	251.496	UPDATE warehouse SET w_ytd = w_ytd + $1 WHERE w_id = $2
2	121	17,440.646	144.138	UPDATE district SET d_ytd = d_ytd + $1 WHERE d_w_id = $2 AND d_id = $3
3	123	16,534.022	134.423	SELECT d_next_o_id, d_tax FROM district WHERE d_id = $1 AND d_w_id = $2 FOR UPDATE
4	14	393.738	28.124	SELECT COUNT(DISTINCT (s_i_id)) AS stock_count FROM order_line, stock WHERE ol_w_id = $1 AND ol_d_id = $2 AND ol_o_id < $3 AND ol_o_id >= $4 - 20 AND s_w_id = $5 AND s_i_id = ol_i_id AND s_quantity < $6
5	101	2,062.8	20.424	DELETE FROM new_order WHERE no_d_id = $1 AND no_w_id = $2 AND no_o_id = $3
6	9	57.296	6.366	SELECT MAX(o_id) AS maxorderid FROM oorder WHERE o_w_id = $1 AND o_d_id = $2 AND o_c_id = $3
7	8	24.826	3.103	SELECT count(*) AS namecnt FROM customer WHERE c_last = $1 AND c_d_id = $2 AND c_w_id = $3
8	123	381.49	3.102	SELECT c_discount, c_last, c_credit, w_tax FROM customer, warehouse WHERE w_id = $1 AND w_id = c_w_id AND c_d_id = $2 AND c_id = $3
9	14	39.89	2.849	SELECT d_next_o_id FROM district WHERE d_w_id = $1 AND d_id = $2
10	101	186.808	1.85	SELECT no_o_id FROM new_order WHERE no_d_id = $1 AND no_w_id = $2 ORDER BY no_o_id ASC
11	100	163.813	1.638	UPDATE customer SET c_balance = c_balance + $1, c_delivery_cnt = c_delivery_cnt + 1 WHERE c_id = $2 AND c_d_id = $3 AND c_w_id = $4
12	1244	1,971.232	1.585	SELECT s_quantity, s_data, s_dist_01, s_dist_02, s_dist_03, s_dist_04, s_dist_05, s_dist_06, s_dist_07, s_dist_08, s_dist_09, s_dist_10 FROM stock WHERE s_i_id = $1 AND s_w_id = $2 FOR UPDATE
13	9	14.195	1.577	SELECT ol_i_id, ol_supply_w_id, ol_quantity, ol_amount, ol_delivery_d FROM order_line WHERE ol_o_id = $1 AND ol_d_id =$2 AND ol_w_id = $3

Figure 4: Sample Query Profiler output

Asynchronous Pre-Fetch

Asynchronous Pre-Fetch is a feature in Linux versions of Postgres Plus Advanced Server and community PostgreSQL that allows the database server to take full advantage of Redundant Array of Inexpensive Disks (RAID) hardware. However, as will be discussed later in this section, EnterpriseDB has further enhanced the capabilities of Asynchronous Pre-Fetch for Postgres Plus Advanced Server.

When using RAID hardware, database data can be allocated on multiple disk drives using a number of different techniques:

- *Striping* is when the RAID controller splits the data into multiple chunks and writes each chunk to a different disk drive.
- *Mirroring* is when the RAID controller writes an exact copy of the data to each disk drive.

A combination of striping and mirroring may also be employed.

Using RAID hardware with Asynchronous Pre-Fetch is especially beneficial in the following situations:

- Data warehouse applications
- Extract, transform, and load (ETL) applications

- Applications typified by a small number of complex queries

Without Asynchronous Pre-Fetch, disk I/O is serialized – that is only one I/O operation occurs at a time keeping only one disk drive in the array active while the others remain idle.

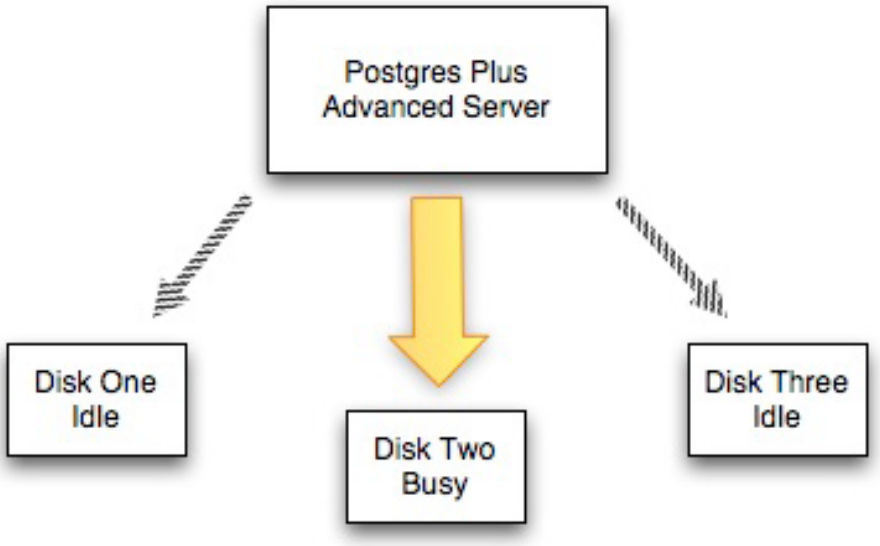


Figure 5: Without Asynchronous Pre-Fetch

When Asynchronous Pre-Fetch is activated, multiple disk I/O requests are run in parallel, thus allowing multiple disk drives to work concurrently and thereby reducing total disk access time. This benefits performance regardless of whether the striping or mirroring technique is used.

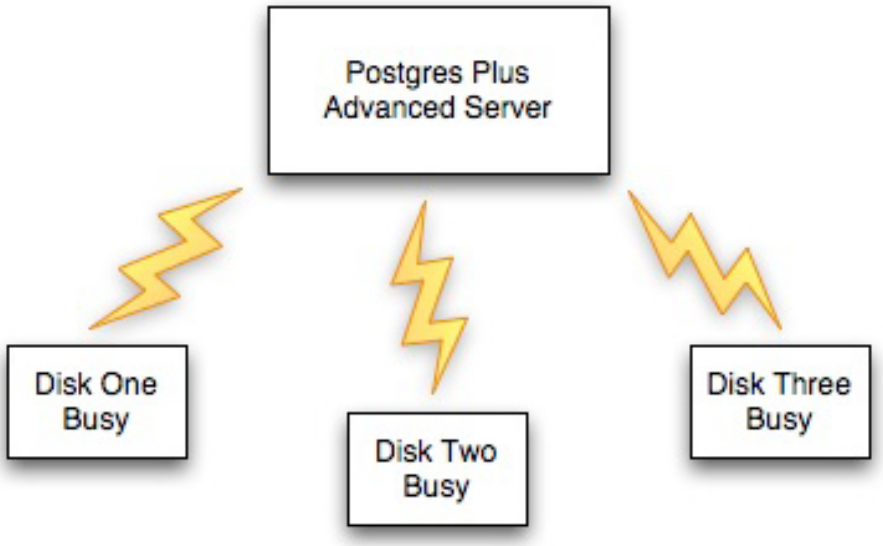


Figure 6: With Asynchronous Pre-Fetch

Asynchronous Pre-Fetch boosts performance when the query optimizer chooses a bitmap scan to access data. Queries where logical AND/OR operations can be applied to the indexes will have the index data converted to bitmaps in memory on which the operations are applied to determine the selected rows. The use of multiple drives speeds up the access of the selected rows.

The version of Asynchronous Pre-Fetch just described was contributed to the community PostgreSQL version 8.4 by EnterpriseDB. However, Postgres Plus Advanced Server now contains an enhanced version of Asynchronous Pre-Fetch that provides the same multiple, concurrent drive access capability to the usage of regular index scans in addition to bitmap index scans.

EnterpriseDB Postgres Plus Advanced Server with Asynchronous Pre-Fetch can provide an advantage over PostgreSQL when regular index scans are used in the following situations:

- The index is highly de-clustered – that is the rows are not stored in the same order as the index.
- The index is narrow (one or a small number of columns comprise the index).
- A large number of rows are expected to be retrieved.

Management

Postgres Plus Advanced Server includes a variety of database management tools not available in the community edition of PostgreSQL that can make maintenance dramatically easier, provide performance enhancing insights into internal processing in the database, keep the database software up to date, and assist in meeting complex government compliance regulations.

DBA Management Server

The DBA Management Server allows database administrators to monitor real-time database performance, view statistics, and to identify configuration issues for an unlimited number of Postgres Plus and/or Postgres Plus Advanced Server databases. It also provides many prepared reports that can be generated in both HTML and PDF formats.

Figure 7: The browser based DBA Management Server

The DBA Management Server has features to aide administrators in maintaining their production Postgres instances. One of the features lacking in PostgreSQL is job scheduling to perform routine maintenance.

While simple operating system specific scheduling tools such as *cron* may work adequately for simple development and testing purposes, their "one-off" batch nature presents consistency problems when deployed in enterprise infrastructures. The job scheduler in the DBA Management Server provides a consistent browser based interface across all operating systems allowing administrators to easily view all Postgres maintenance jobs from a single console.

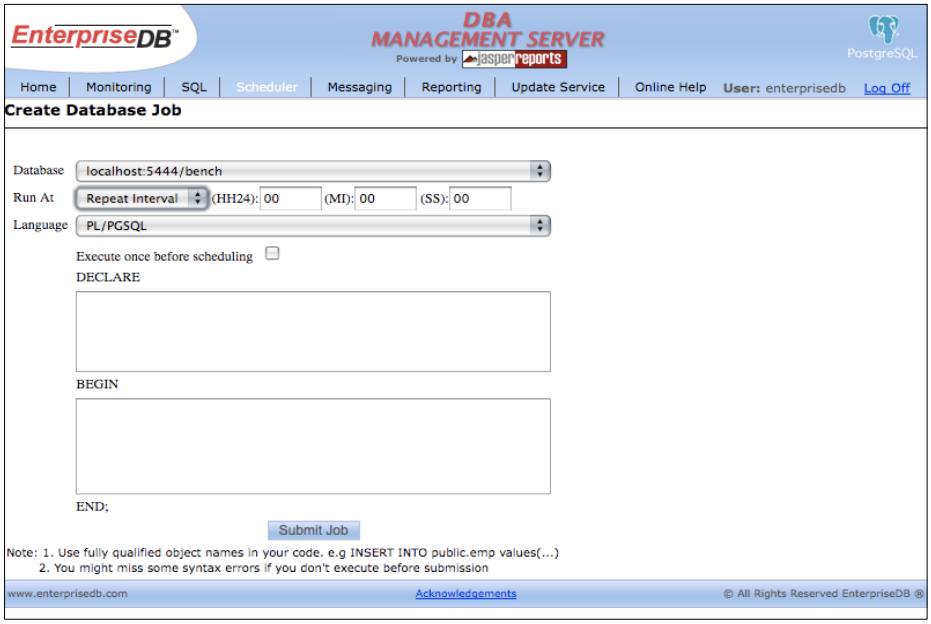


Figure 8: The DBA Management Server Job Scheduler

Other tasks handled by the DBA Management Server include:

- Monitoring user activity, lock status, and buffer caches
- Performing Query Profiling as discussed previously
- Issuing interactive SQL queries
- Checking and scheduling the automatic update notification service
- Running a variety of statistics, meta data, procedural logic, and security reports

DBA Monitoring Console

The DBA Monitoring Console provides DBAs a quick, easy-to-use, real-time dashboard into the system usage consumed by one or more Postgres Plus and/or Postgres Plus Advanced Server database servers. Graphs are presented for the following information:

- Percentage of CPU used
- Percentage of memory used
- Number of data blocks read from disk
- Number of data blocks read from the buffer cache

Figure 9: The DBA Monitoring Console

The graphs can be displayed on one screen as shown above, or they can be displayed individually. A preference can be set for each graph to display the information as a line or as a bar chart.

A chart can also be displayed showing the individual processes connected to each database in the server and its current query.

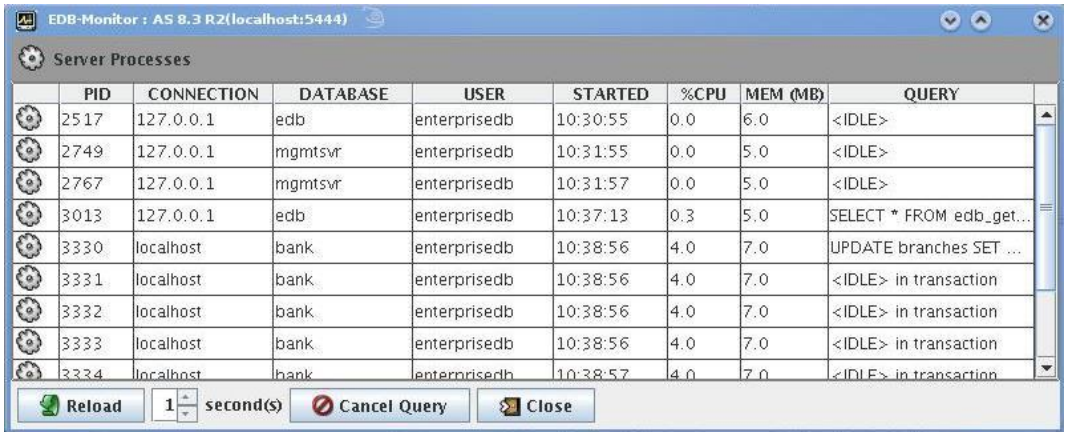


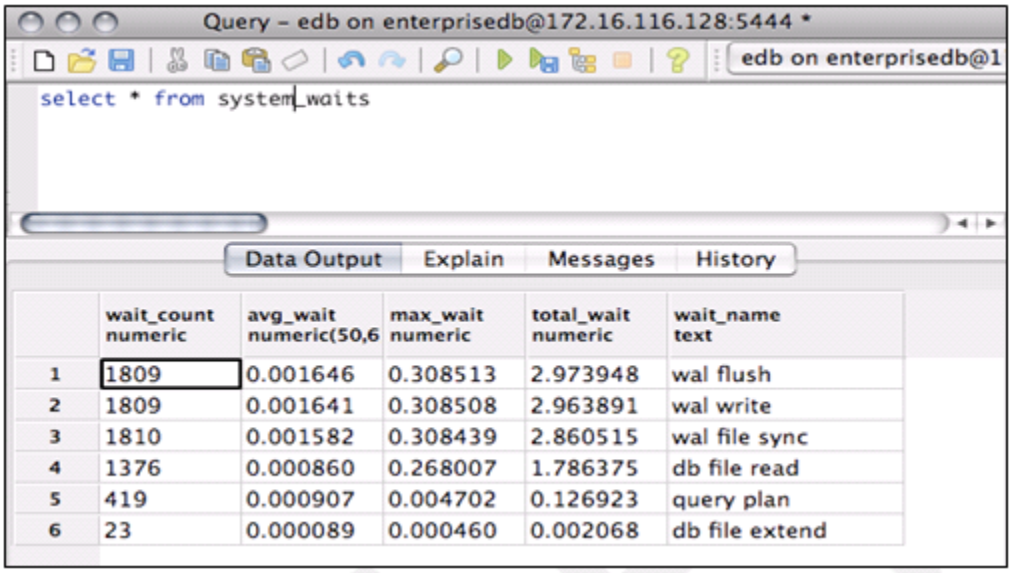
Figure 10: The DBA Monitoring Console Server Processes

The rate at which the information is refreshed can be adjusted for the server processes chart as well as for the four graphs.

Dynamic Runtime Instrumentation Tools Architecture (DRITA)

DRITA allows DBAs to query new catalog views containing runtime statistics while the database is under load. This information can be used to determine what *wait* events are affecting the performance of individual sessions or the system as a whole. The number of times an event occurs as well as the time spent waiting is collected. By viewing these waits in descending order as a percentage, it is possible to see, at a glance, which events are impacting performance and then take corrective action.

The graphic below shows an example of the system load during a “pgbench” run. The results in this example show that the majority of the work being performed by the database is waiting for the WAL (Write Ahead Log). In our simplistic example, this is the logical conclusion because “pgbench” is a heavy transactional test, but in more complex scenarios typical of production business applications, this amount of detail is essential in pinpointing bottlenecks.



	wait_count numeric	avg_wait numeric(50,6	max_wait numeric	total_wait numeric	wait_name text
1	1809	0.001646	0.308513	2.973948	wal flush
2	1809	0.001641	0.308508	2.963891	wal write
3	1810	0.001582	0.308439	2.860515	wal file sync
4	1376	0.000860	0.268007	1.786375	db file read
5	419	0.000907	0.004702	0.126923	query plan
6	23	0.000089	0.000460	0.002068	db file extend

Figure 11: DRITA System Waits View

The Dynamic Runtime Instrumentation can also be used for forensics of past performance problems. The history of *wait* statistics is maintained so administrators can diagnose issues that may be reported well after the period of poor performance has passed (a common situation confronting enterprise database administrators).

For example, a user reports at an end of day status meeting that an application was running slow between 1:00 and 1:30. The DBA can track down the issue through DRITA's extensive views and functions. Consider the troubleshooting sequence below, which further illustrates how DRITA can be used to pinpoint the cause of performance problems.

STEP 1: Examining the *edb$snap* table which contains all of the information about historical performance measures. The DBA constructs a query that returns a range of *snap_id* values for the time period under investigation. This query is used to determine which database session may be causing the problem. In this example, session 3256 appears to be problematic.

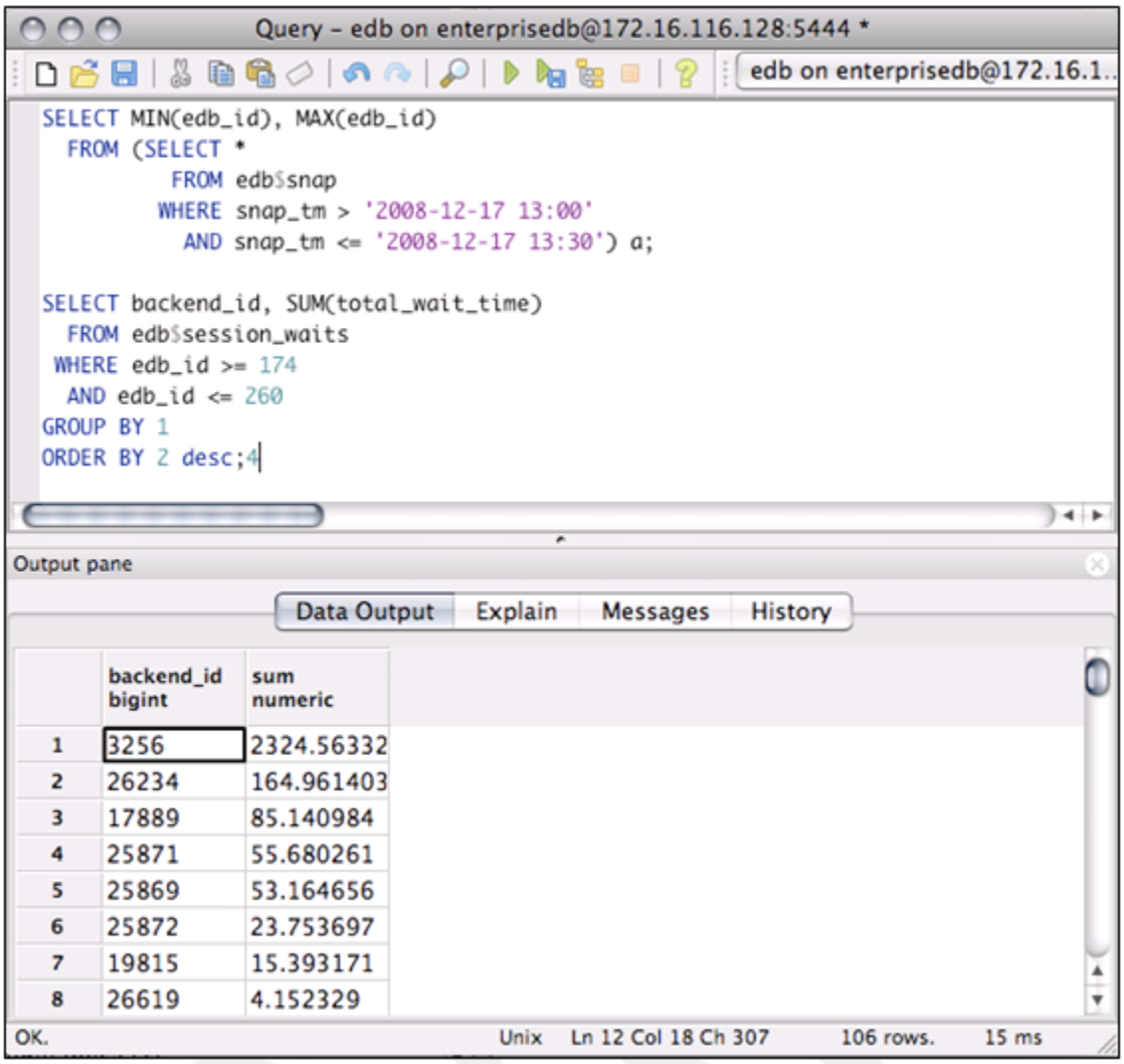


Figure 12: DRITA Output from edb$snap

STEP 2: Examining user activity details. Armed with the id of the potential problem session, the DBA can then drill down for details, find the problem and take corrective action.

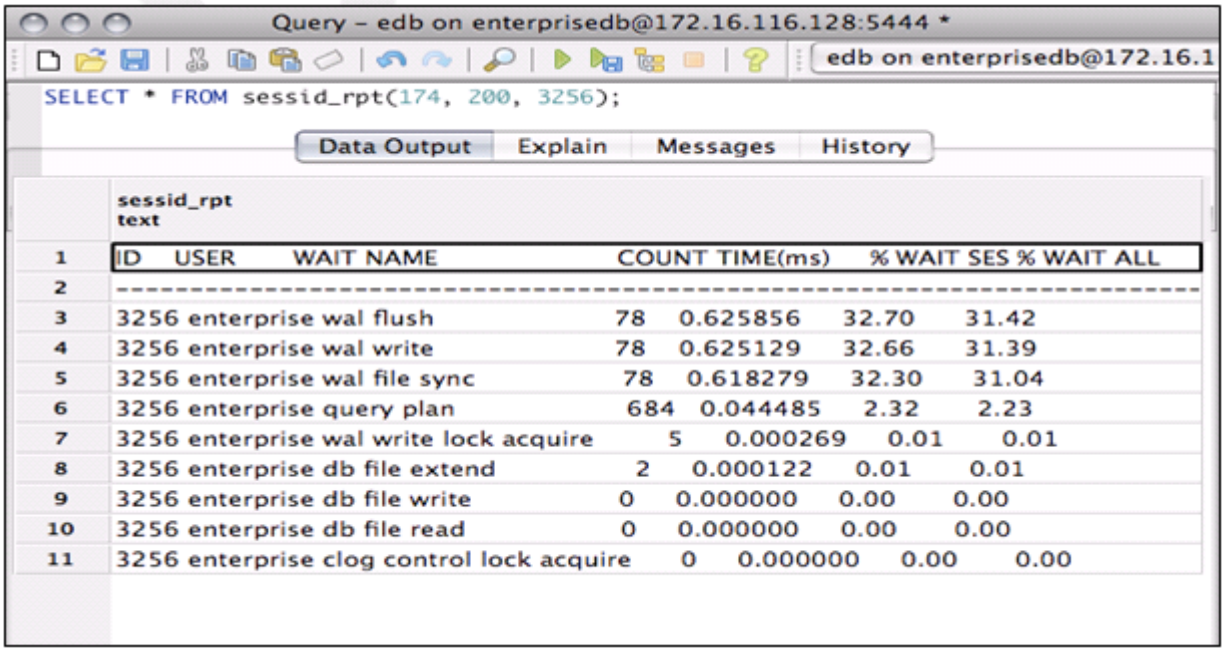


Figure 13: DRITA output from sessid_rpt

Update Service

The EnterpriseDB Update Service is a feature of the DBA Management Server. The browser based console provides a number of crucial functions for database administrators including:

- Configurable automatic notification of pending updates and patches for Postgres and installed companion products
- Comprehensive listing of installed companion products and versions
- Viewing of the Update Logs
- Manual checking for database updates and patches
- Easy application of database patches through a web based console

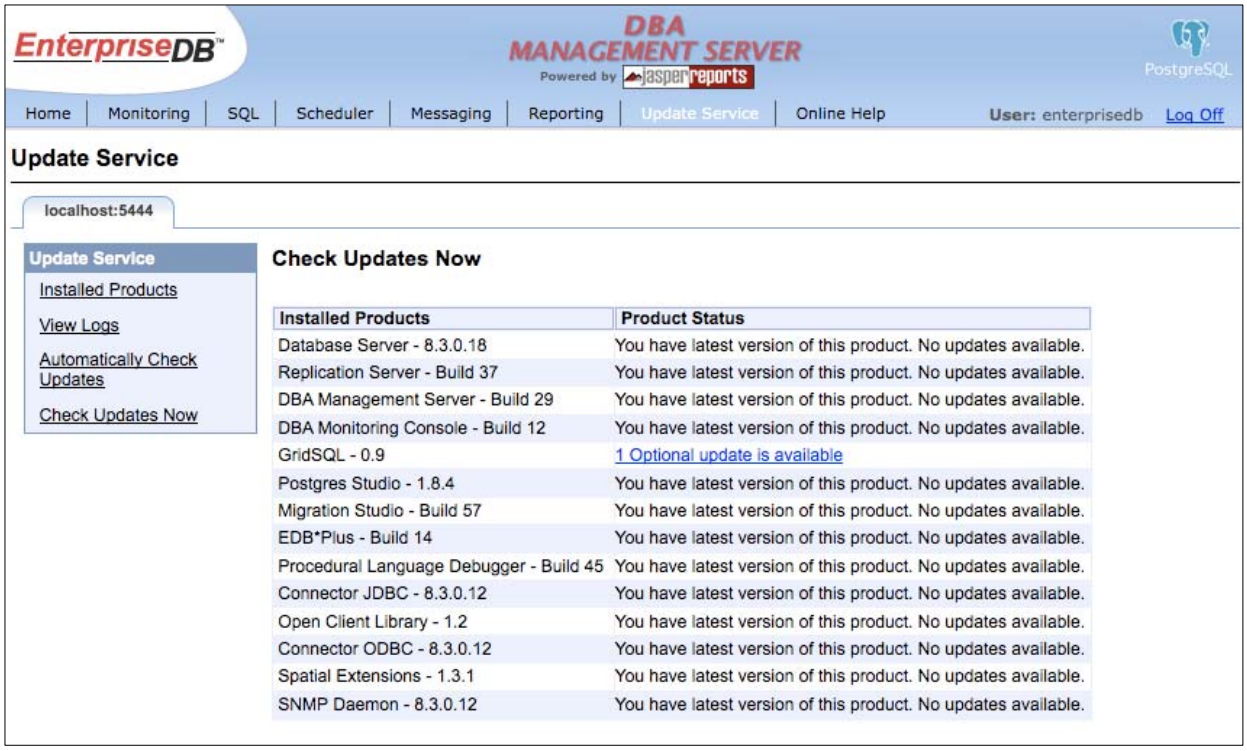


Figure 14: DBA Management Server Update Service

Audit Logging

Postgres Plus Advanced Server provides the capability to produce audit reports. Database auditing allows database administrators, security administrators, auditors, and operators to track and analyze database activities in support of complex government auditing requirements.

These audited activities include database access and usage along with data creation, change, or deletion. The auditing system is based on configuration parameters defined in the configuration file.

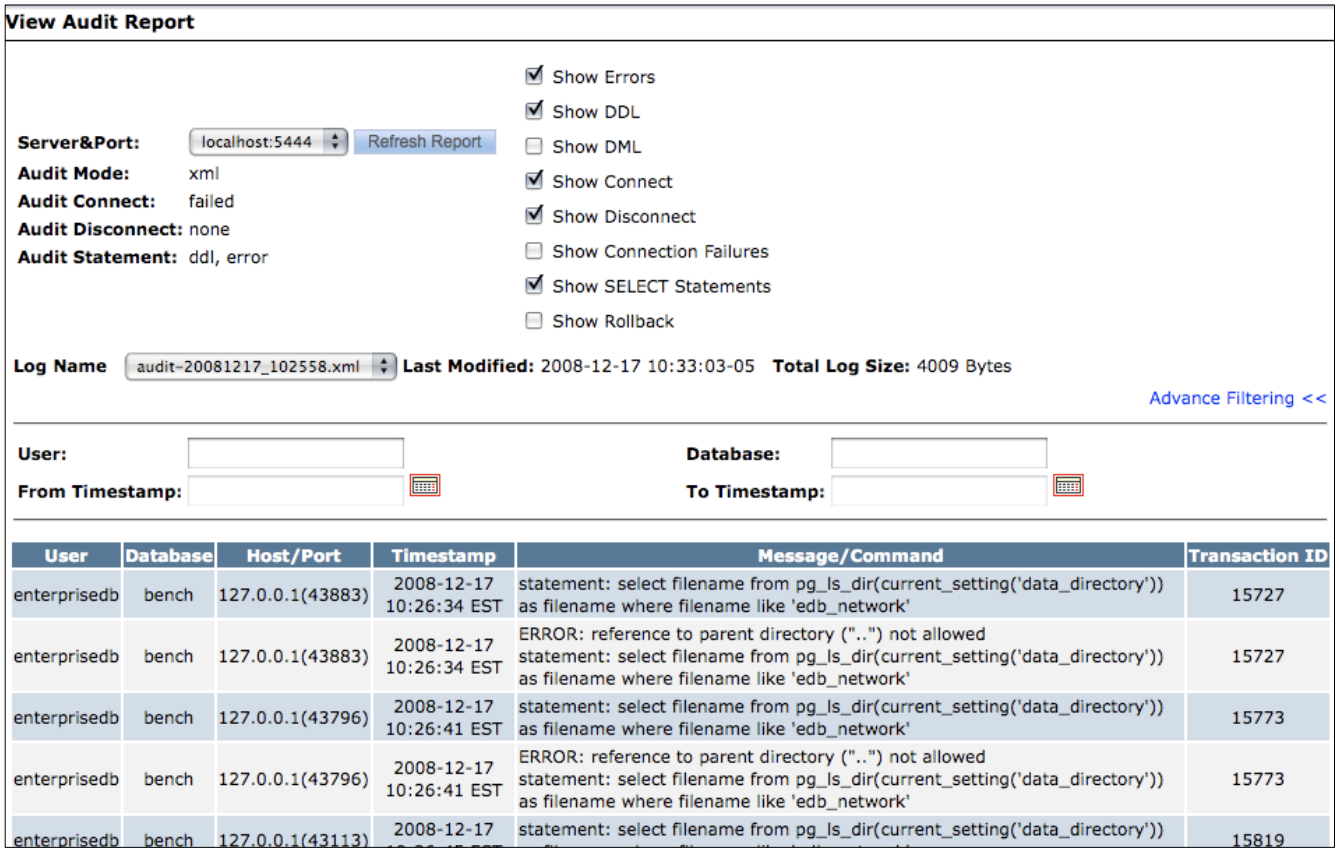


Figure 15: DBA Management Server Auditing Report

The following audit features are also available to help DBAs utilize, tune, and manage the auditing system:

- Two formats for report output: xml or csv
- Audit files can be rotated on a regular basis based on:
  - the day of the week
  - a specified maximum size of the audit file
  - a specified interval of time
- Actual and attempted connects and disconnects to and from the database can be audited
- Specific types of SQL statements (e.g. those resulting in errors, all DDL, or all DML).

Database Migration

Where PostgreSQL is prepared to migrate MySQL databases in need of more robust OLTP support, Postgres Plus Advanced Server can also migrate Sybase, Microsoft SQL Server, and Oracle databases. Many organizations today feel they are paying inflated license fees for non-

critical or adjacent applications in their data centers. And because these applications can be very costly to re-write and port over to another database system, users often feel stuck with no good alternatives.

EnterpriseDB has made database compatibility a key focus of its ongoing enhancements included in Postgres Plus Advanced Server. Currently, the Migration Studio in Postgres Plus Advanced Server can migrate objects according to the following table:

Object	Sybase	Microsoft	Oracle
Schema	✓	✓	✓
Data	✓	✓	✓
Constraints			✓
Sequences			✓
Synonyms			✓
Indexes	✓	✓	✓
Packages			✓
Stored Procedures			✓
Triggers			✓
Functions			✓
Views		✓	✓
Users, Roles			✓
Database Links			✓
List-Partitioned Tables			✓
Range-Partitioned Tables			✓

Table 2: Database Migration

Usability

One of the defining characteristics of commercial software is usability. Products, indeed even companies, have grown to require ease of use across a range of activities including software acquisition, installation, configuration, use, maintenance, and integration with other complementary products.

A persistent criticism of many open source software products is specifically how difficult they are to use and manage along the dimensions mentioned above. Usability can be especially problematic with the proliferation of related, complementary open source projects.

For someone initially investigating open source alternatives, or even an open source veteran under a tight timeline, finding the right project(s), downloading the source, compiling it, configuring it, integrating with other

components, and testing the whole result can be massively time consuming and error prone.

With Postgres Plus Advanced Server, EnterpriseDB has gone to great lengths to make it the easiest Postgres based database available. Utilizing a customer base composed of many global enterprises, EnterpriseDB has cataloged and consolidated those additional database features expected by most database professionals and organizations today, and made them available in a single, integrated and easy to use product.

Enterprise Class Packaging

Postgres Plus Advanced Server is built upon the enterprise class features of the standard PostgreSQL source base and adds to that foundation multiple additional open source modules that can be classified into four categories: Extensions, Security, Procedural Languages (Postgres supports over 10 programming languages), and Contributing modules.

The tables below summarize all the component projects that comprise Postgres Plus Advanced Server:

Category	Feature	Description
Extensions	Slony	Slony is a trigger based asynchronous master to many slaves replication system.
	pgAdmin III	pgAdmin III is a graphical administration and development platform for Postgres.
	PostGIS	PostGIS is an extension to Postgres enabling robust spatial database capabilities.
Security	SSL	Postgres Plus has native support for using SSL connections to encrypt client/server communications for increased security.
	Kerberos	Postgres Plus supports Kerberos version 5 as an authentication method.
Procedural Languages	PL/pgSQL	PL/pgSQL is a loadable procedural language that adds control structures to the SQL language.
	PL/Perl	PL/Perl is a loadable procedural language that enables users to write Postgres Plus functions in the Perl programming language.
	PL/TCL	PL/TCL is a loadable procedural language that enables users to write Postgres Plus functions in the TCL programming language.

	PL/Java	PL/Java is a loadable procedural language that enables users to write Postgres Plus functions in the Java programming language.
Contrib Modules	citext	The citext provides a case-insensitive character string type.
	oid2name	oid2name is a utility program that allows administrators to examine the file structure of Postgres Plus.
	pgbench	pgbench is a program for running benchmark tests on Postgres Plus.
	adminpack	adminpack provides a number of support functions used by administration and management tools to provide additional functionality, such as remote management.
	chkpass	chkpass implements a data type used for storing encrypted passwords.
	cube	cube implements a data type to represent multi-dimensional cubes.
	dblink	dblink is a module that allows connections to other Postgres databases from within a database session.
	earthdistance	The earthdistance module provides two different approaches to calculating great circle distances on the surface of the Earth.
	fuzzystrmatch	The fuzzystrmatch module provides several functions to determine similarities and distance between strings.
	hstore	hstore implements a data type for storing sets of (key,value) pairs within a single Postgres data field.
	intagg	intagg provides an integer aggregator and an enumerator.
	isn	isn implements data types for international product numbering standards
	lo	lo provides support for managing Large Objects
	ltree	ltree implements a data type for representing labels of data stored in a hierarchical tree-like structure
	pg_buffercache	pg_buffercache allows for examining the shared buffer cache in real time.
	pg_freespacemap	pg_freespacemap allows for examining the free space map
	pg_standby	pg_standby allows for the creation of a "warm standby" database server.
	pg_trgm	pg_trgm module implements functions and operators for determining the similarity of text.
	pgcrypto	pgcrypto implements cryptographic functions for Postgres
	pgrowlocks	pgrowlocks provides a function to show row locking information for a specified table.

pgstattuple	pgstattuple implements various functions to obtain tuple-level statistics.
seg	seg implements a data type for representing line segments.
tablefunc	tablefunc provides various functions that return tables.
tsearch2	tsearch2 implements full text indexing.
uuid-ossp	uuid-ossp provides functions to generate universally unique identifiers.
xml2	xml2 provides XPath querying and XSLT functionality

Table 3: Postgres Plus Advanced Server Add-On Modules

All the above features are integrated into a simple intuitive installer and certified through a repeatable and extensive quality assurance process. This ensures that all pieces encompassing many open source projects function as a single, coherent, general, enterprise class database system. Finally, the automatic update notification system delivered with Postgres Plus Advanced Server ensures that all modules are kept up to date and in synch.

PL Debugger

The PL Debugger gives developers and DBAs the ability to test and debug Postgres Plus server-side programs using a graphical, dynamic interface. The PL Debugger can be used on SPL stored procedures, functions, and triggers as well as PL/pgSQL functions. Users can follow code paths, set breakpoints, edit variables, and step into and over code.

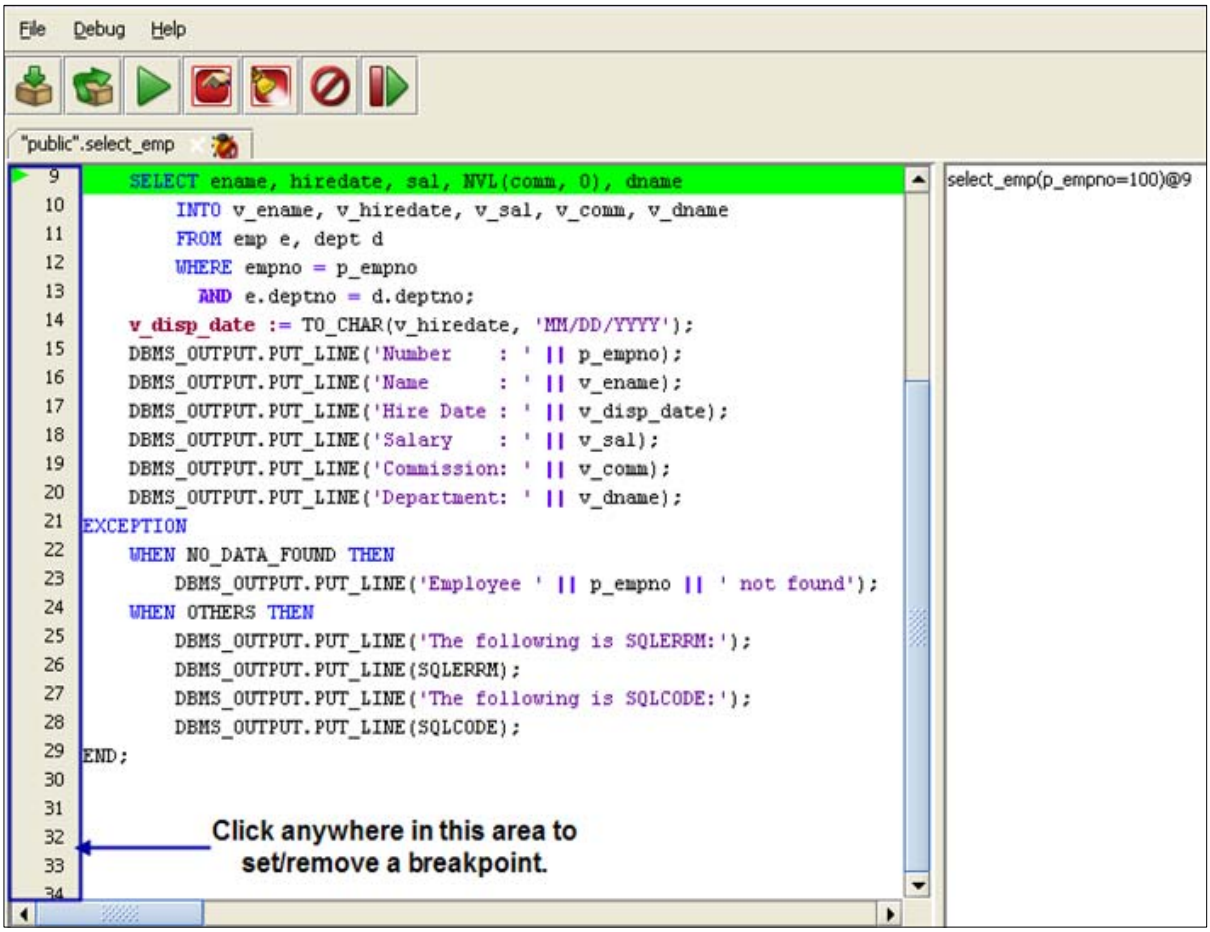


Figure 16: The PL Debugger

The PL Debugger is integrated into Postgres Studio, the cross platform database management and development environment for Postgres databases. One of the real strengths of the PL Debugger is the ability to test server side logic, such as triggers, in the context of the actual application that executes them.

For example, a global breakpoint can be set in some trigger code related to employee salary increases. The developer can then start the application, make a salary change, and have the trigger code halt at the

appropriate breakpoint. From there the developer can step through the trigger logic and watch how the data is treated.

This type of server side debugging can be extremely beneficial in resolving problems where more than one application utilizes common database code. Developers can debug trigger code in the context of each referencing application.

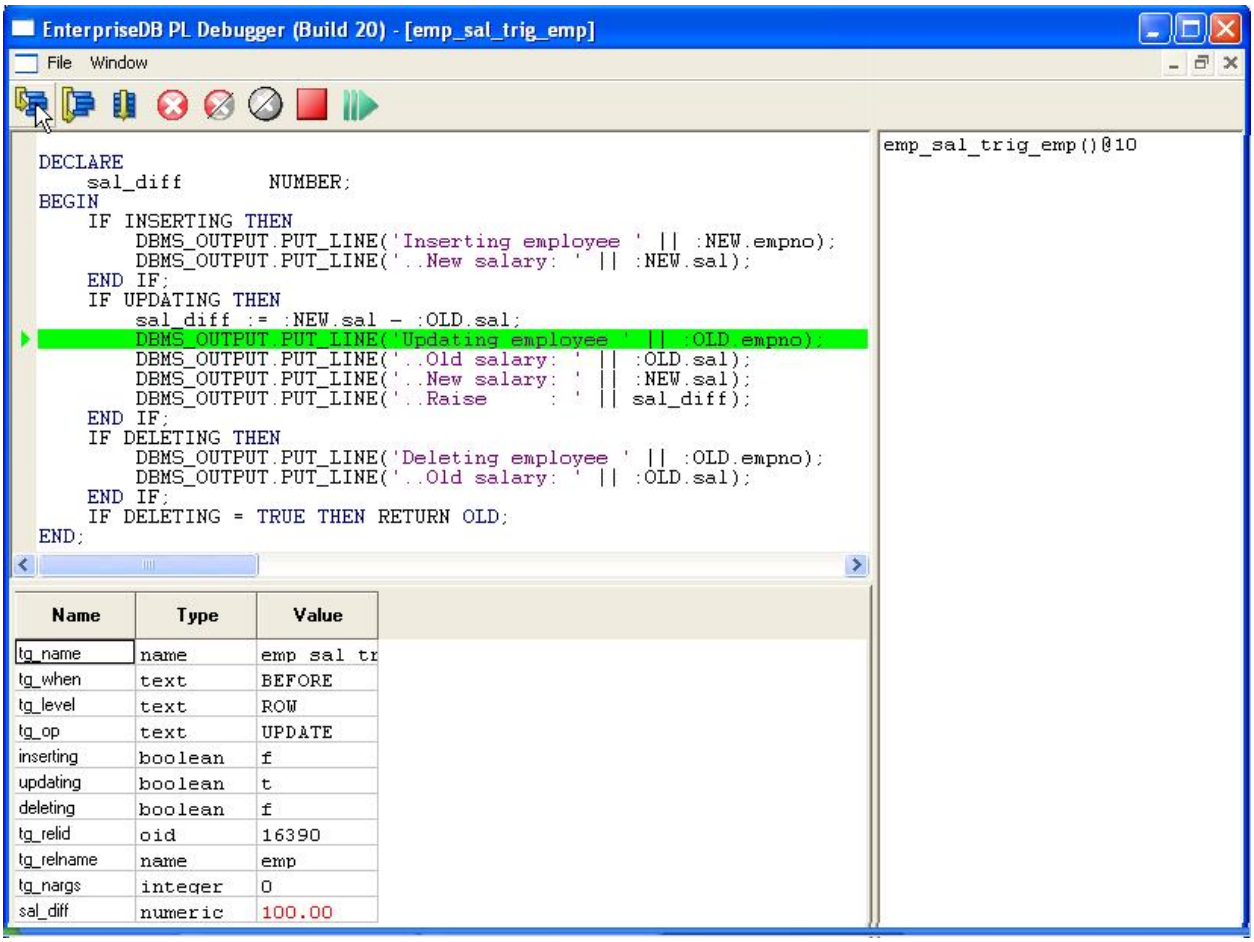


Figure 17: Server side trigger debugging in the application context

Transaction Error Recovery

Within PostgreSQL, an application that produces even a single error causes the active transaction to be placed into a failure state. At that point, the only operation the application can do is to roll back that transaction. These errors can be raised by anything the database flags as an error condition.

One of the most common error scenarios is the handling of duplicate key violations. The functional requirement in many applications is to add a new record to a table but, if its key already exists, then the existing record should be updated. With PostgreSQL, it is necessary to check if a record exists before issuing an insert or update transaction, forcing this operation to consume two statements. With the use of Transaction Error Recovery, the application can issue an insert transaction and then only issue an update transaction if the duplicate key is detected.

The behavior of PostgreSQL in a duplicate key scenario is shown below.

```
ResultSet rs = stmt.executeQuery("SELECT COUNT(*) " +
                        "FROM TELLERS WHERE TID=1000");
rs.next();
if (rs.getInt(0) == 0) {
  stmt.executeUpdate("INSERT INTO TELLERS(TID, BID) " +
          "VALUES(1000, 1)");
} else {
  stmt.executeUpdate("UPDATE TELLERS SET BID=1 " +
          "WHERE TID=1000");
}
```

The behavior of Postgres Plus Advanced Server with Transaction Error Recovery activated is shown below.

```
try{
  stmt.executeUpdate("INSERT INTO TELLERS(TID, BID) " +
          "VALUES(1000, 1)");
} catch (Exception e) {
  stmt.executeUpdate("UPDATE TELLERS SET BID=1 " +
          "WHERE TID=1000");
}
```

Scalability

Enterprise information architectures are growing at ever faster rates, creating significant data management problems. Postgres Plus Advanced Server offers cost effective solutions for scaling a wide range of database infrastructures, from the consolidation of existing database systems to the communication between multiple independent database systems to the exponential growth of Web 2.0-style applications.

GridSQL

GridSQL is a horizontal scaling solution that allows organizations to rapidly expand and manage their database infrastructures using inexpensive commodity hardware.

GridSQL is an open source project sponsored by EnterpriseDB and fully integrated into Postgres Plus Advanced Server. It enables organizations to easily meet complex data warehousing and business intelligence challenges using grid technology that implements a "shared-nothing", distributed data architecture.

GridSQL's intelligent partitioning engine allows data to be distributed across all nodes in a configurable manner, and then parallelizes query execution across the distributed infrastructure. GridSQL clusters together independent database nodes at the database layer and presents them to applications as a unified, virtual database. Existing applications run unchanged and application developers can create new applications as if they were accessing a single database instance on a single server.

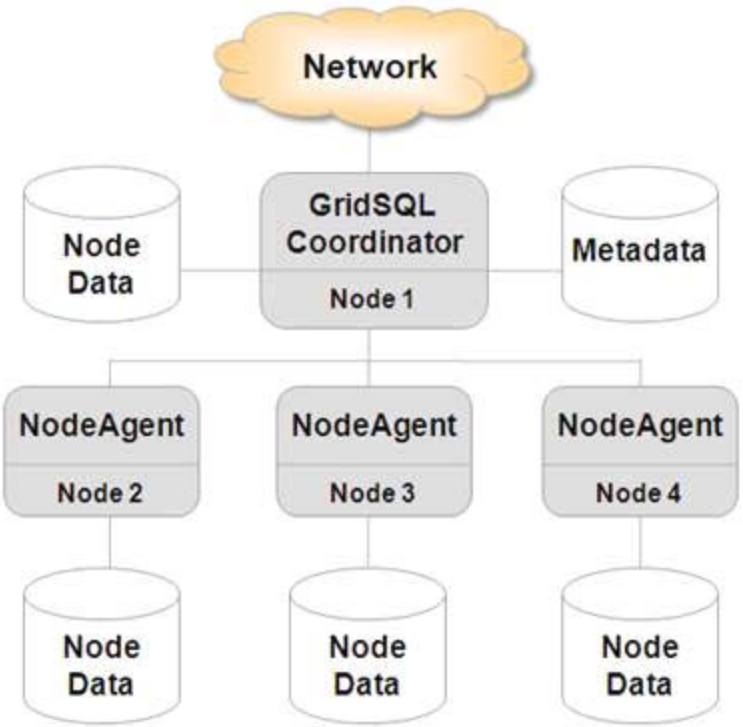


Figure 18: GridSQL Architecture

Using GridSQL, database performance improves in a linear fashion as additional commodity servers are added to the grid. To simplify management, GridSQL includes a robust suite of graphical management and monitoring tools that enable DBAs to monitor the entire grid from a single console.

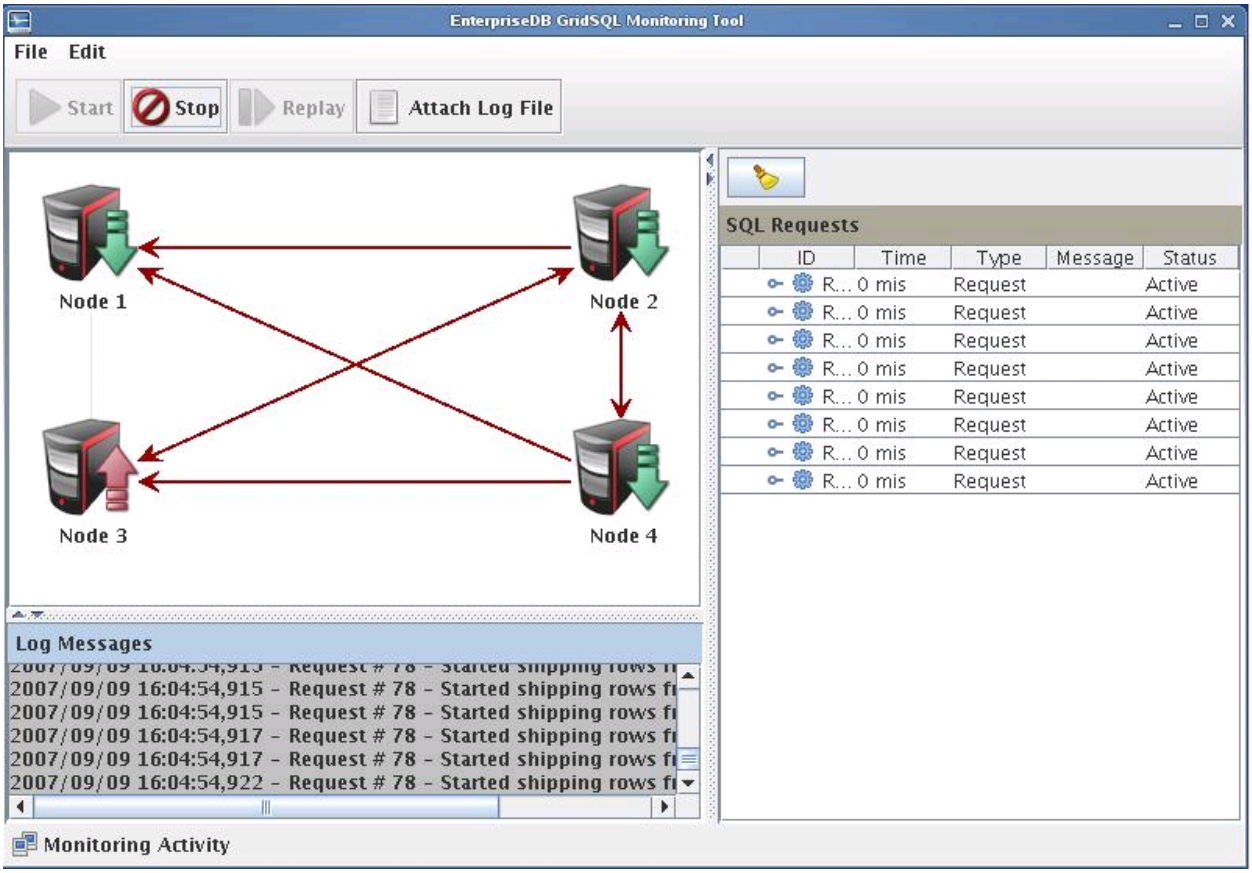


Figure 19: GridSQL Management Console

Postgres Plus Advanced Server includes the Grid Monitoring Tool that allows the DBA to monitor and tune the entire grid. In addition, it visually depicts the traffic among and within the nodes so that the entire grid can be balanced. The Postgres Plus Advanced Server version of GridSQL also makes provisioning a snap, allowing nodes to be easily added to or removed from the grid.

Infinite Cache

One of the constraints on database performance is the amount of memory available in the database server for the shared buffer cache. All requested data must first be read into the shared buffer cache before it can be utilized. Once in the shared buffer cache, access to data is fast compared to the time it takes to read the data from disk.

Typically, though, there comes a point where the shared buffer cache is full and any additional requested data from disk must overwrite some data

already in the buffer cache. This overwritten data may be required again at some point where upon, the data must be read in from disk again. This constant competition for a limited amount of memory is what can become an upper bound on performance.

Thus, it follows that if there was an infinite amount of memory available, performance can reach a more optimum level as there would be no need to re-read data from disk once it already has a place in memory.

EnterpriseDB's Infinite Cache for Postgres Plus Advanced Server is built on this concept. Available on Linux systems, Infinite Cache allows Postgres Plus Advanced Server to utilize memory from networked, commodity computers as a secondary memory cache (cache servers).

This provides a "cache blade" architecture using commodity hardware and eliminates the need to upgrade the database server to a more expensive platform such as Unix.

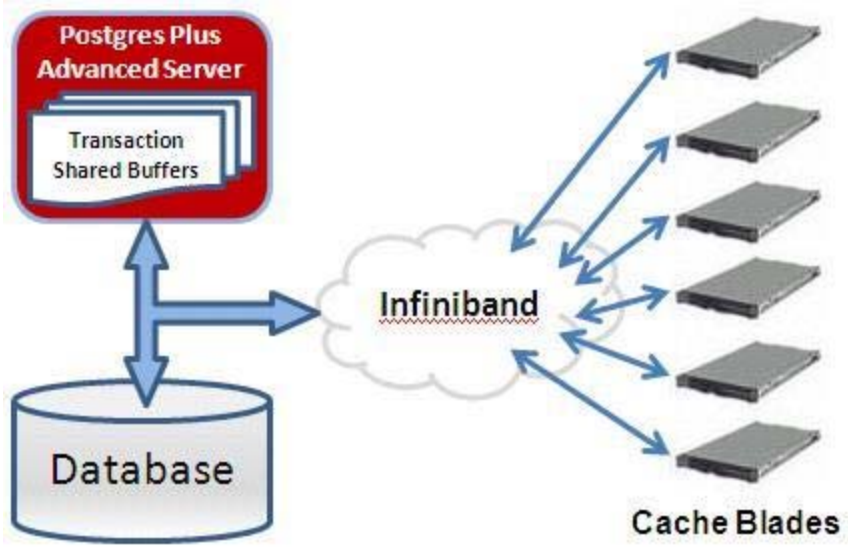


Figure 20: Infinite Cache Architecture

With Infinite Cache in place, when a block of data is read from disk for the first time, it is also stored on a cache server as well as in the shared buffer cache. If at some later point in time, that same data block is requested again, the shared buffer cache is searched first.

If the data block had been evicted from the shared buffer cache, the cache servers are searched next instead of going to disk. Since the secondary cache can be significantly larger than the shared buffer cache, chances are that the data block still resides on a cache server, and the data block can be retrieved from there, which still provides quicker retrieval than disk

access. Only if the data is not on any of the cache servers, is the disk read for the data.

Note that this all happens transparent to the application – no program coding changes are required to take advantage of Infinite Cache.

To further increase the capacity of the secondary cache, Infinite Cache provides the capability to compress the data to varying degrees before it is placed on a cache server. Compression allows more data to fit in the secondary cache and also reduces the amount of bandwidth required to transmit the data over the network.

Infinite Cache also allows you to "warm" the cache servers – that is selected tables, with or without their indexes, can be pre-loaded into the secondary cache before the application starts. This allows the database server to immediately take advantage of the faster memory access over disk reads if it is known in advance what tables the application will use.

Usage statistics can be obtained from each cache server to analyze performance and efficiency.

```
# edb-icache-tool 192.168.23.85:11211 stats

Field                 Value
bytes                 1051681421
bytes_read            1410538244
bytes_written         42544414583
cmd_get               5139685
cmd_set               126588
connection_structures 104
curr_connections      4
curr_items            126588
evictions             0
get_hits              5139530
get_misses            155
limit_maxbytes        1073741824
pid                   3047
pointer_size          32
rusage_system         109.077417
rusage_user           21.423743
threads               1
time                  1242367107
total_connections     115
total_items           126588
uptime                1095
version               1.2.6
```

The following graph illustrates the dramatic performance increase that can be realized on Postgres Plus Advanced Server with Infinite Cache enabled as compared with running the same test with Infinite Cache turned off.

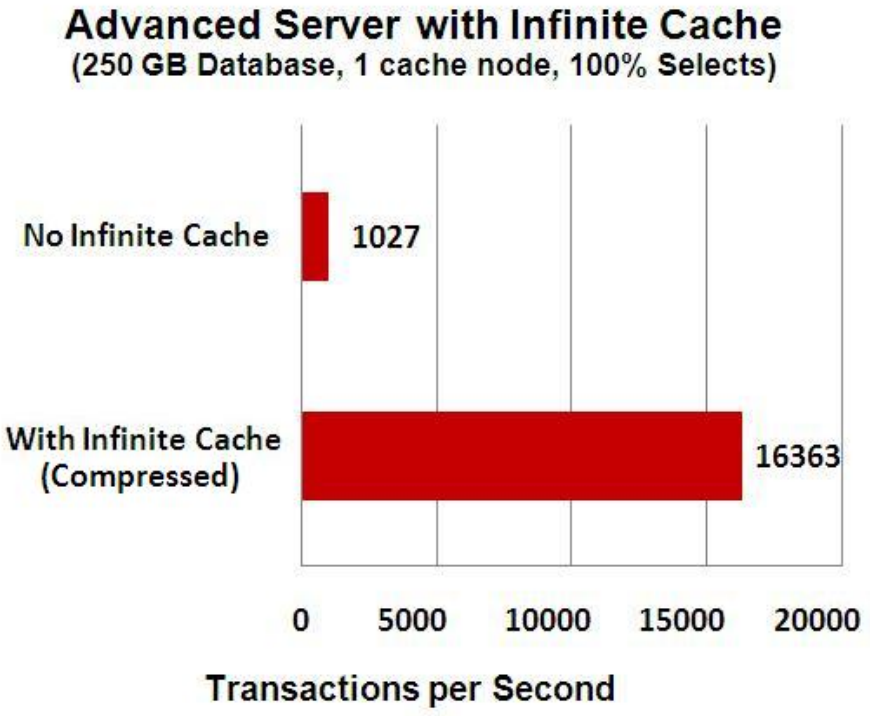


Figure 21: Infinite Cache vs. No Infinite Cache

Since Infinite Cache keeps as much previously read data in the secondary cache as possible, the greatest performance advantage is gained for applications that repeatedly access the same data. The following graph compares two applications – one which includes some updates (the top pair of bars) and the other which is read-only (the bottom pair of bars). As seen by observing the top bar of the bottom pair, Infinite Cache has its most dramatic benefit for the application performing 100% SELECT statements.

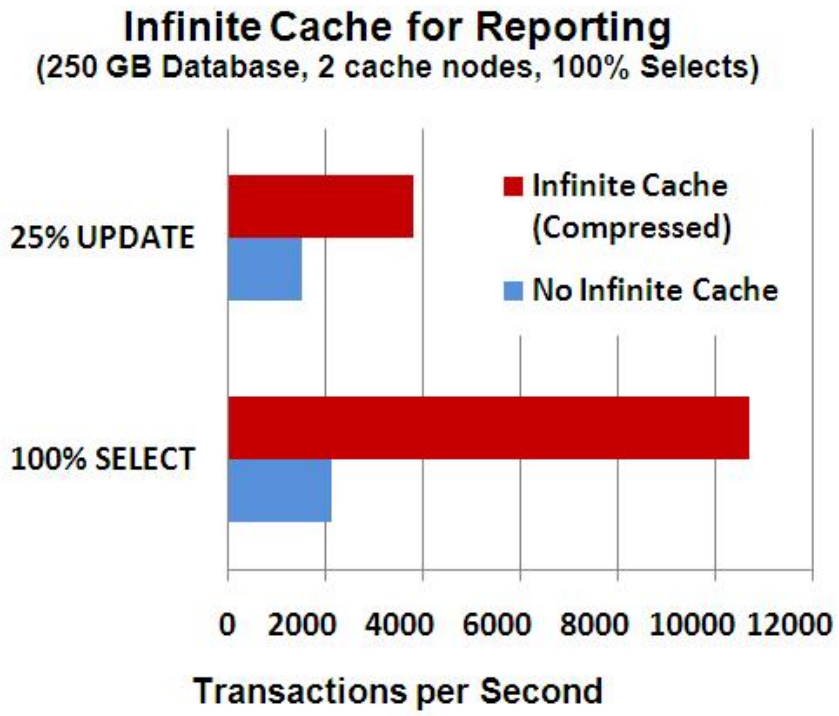


Figure 22: Selects vs. Updates With and Without Infinite Cache

The final graph shows the performance increase gained for both update and read-only applications when using Postgres Plus Advanced Server with Infinite Cache over PostgreSQL. As can be expected, the most dramatic performance gain is in the read-only application.

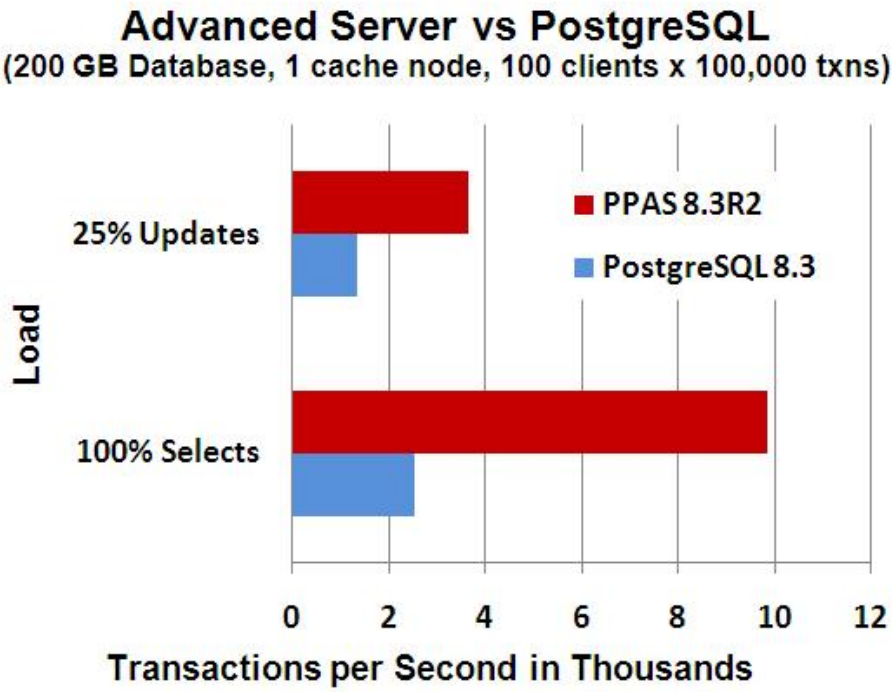


Figure 23: PostgreSQL vs. Postgres Plus Advanced Server With Infinite Cache

In summary, Infinite Cache provides the following features:

- Uses commodity hardware as cache blades to increase memory and limit expansion costs.
- No application programming changes are required.
- Provides for varying degrees of data compression within the cache to allow even more space for cached data and to reduce network bandwidth.
- Pre-loads selected tables and indexes into the cache to provide an immediate performance advantage without a warm-up period.
- Provides tools to obtain comprehensive statistics on cache usage to tune performance.

Database Links

A database link is a reference that defines a communication path from one database server to another. The advantage of database links is that they allow users to access another user's objects in a remote database so that they are bounded by the privilege set of the object's owner.

Database links also form the basis for a homogenous distributed database system. An application can simultaneously access data in several databases in a single distributed environment. For example, a client can issue a single query on a local database and can retrieve joined data from tables on remote databases. For client applications, the location and platform of the databases are transparent.

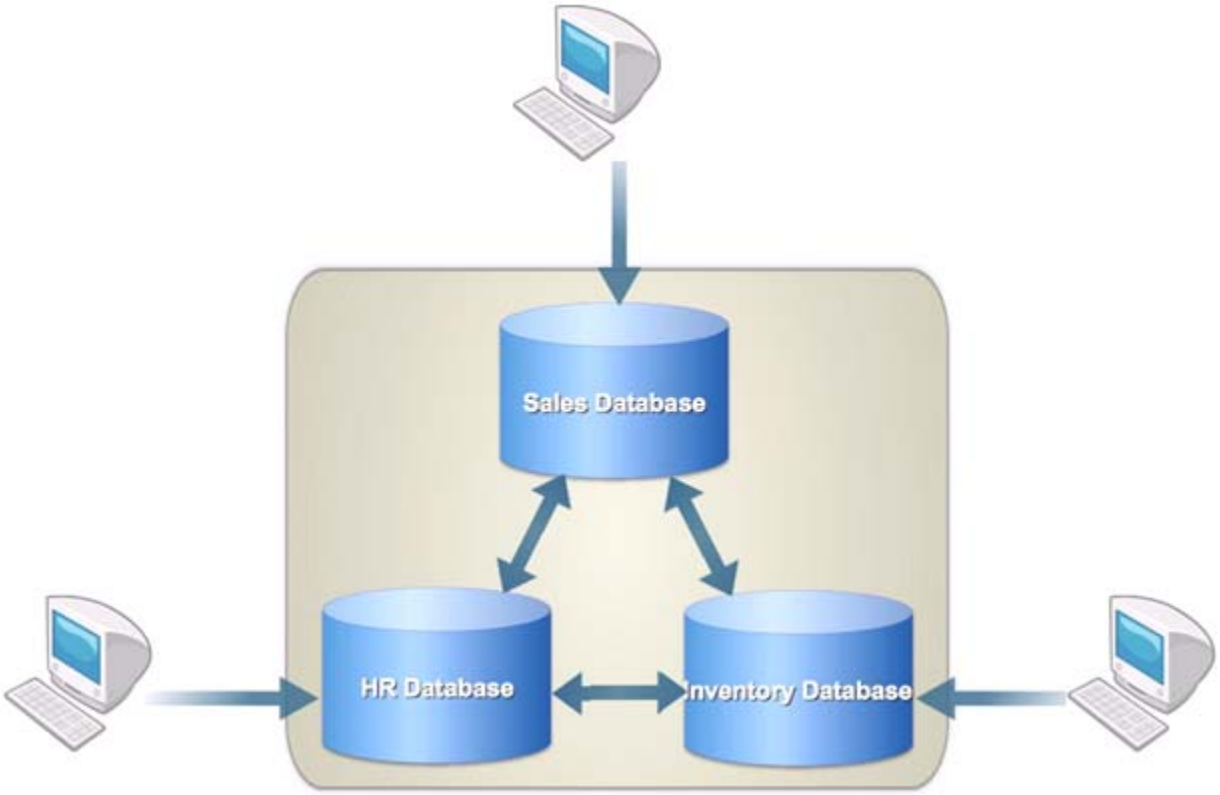


Figure 24: Database Linking

Conclusion

PostgreSQL is the world's most advanced open source database, capable of supporting applications processing billions of transactions a day over a world-wide network.

EnterpriseDB is the leading contributor to the PostgreSQL project and also offers Postgres Plus Advanced Server, an enhanced version of PostgreSQL that includes many enterprise features not found in the community distribution, including ease of installation, management, ease of use, performance, and scalability. In addition, Advanced Server users benefit from EnterpriseDB's full range of solutions including training, 24x7 "follow the sun" technical support and professional services.

Organizations seeking the price/performance of a world-class open source database, along with all of the enterprise class capabilities available in costly proprietary products, have found Postgres Plus Advanced Server to exceed their expectations.

For more information regarding Postgres Plus in your environment, please contact us at: https://www.enterprisedb.com/about/contact_us.do or contact the Sales department at: sales-us@enterprisedb.com (US), sales-intl@enterprisedb.com (Intl), or call +1-732-331-1315, 1-877-377-4352.

About EnterpriseDB

EnterpriseDB is the leading provider of enterprise class products and services based on PostgreSQL, the world's most advanced open source database. The company's Postgres Plus products are ideally suited for transaction-intensive applications requiring superior performance, massive scalability, and compatibility with proprietary database products. Postgres Plus also provides an economical open source alternative or complement to proprietary databases without sacrificing features or quality. EnterpriseDB has offices in North America, Europe, and Asia. The company was founded in 2004 and is headquartered in Westford, MA. For more information, please call +1-732-331-1300 or visit http://www.enterprisedb.com .