Overview

DT2 BDsafe Data Store

Summary

This document gives an overview of the DT2 BDsafe Data Store system.

Contents

0. Definitions and Abbreviations

0.1 Definitions

0.2 Abbreviations

1. Origins

The BDsafe data store system was originally a solution to a problem in the Telco world where very large volumes of event records needed to be,

1. Stored quickly.
2. Retrieved quickly.
3. Stored for long periods.

It also had to be very cost effective.

The BDsafe system architecture assumed,

1. The data to be managed is WORM data.
2. The data is event data and is semi-structured.
3. The record volume is at least 500 Million records per day, and probably multi-Billion records per day.
4. An indexing capability must exist to enable retrieval at the record level.
5. A single record must be retrievable in a few seconds.
6. Storing and querying performance must be constant irrespective of the number of records stored.
7. The BDsafe data store must be easily expandable.

2. Core Capabilities

The purpose of the Bdsafe data store was to enable reliable management of event data.

It can store data for any length of time, given the appropriate resources.

It can create multiple indexes on records.

It can retrieve data at the file level and the record level.

As an indication of its place in the general system world, it can be viewed in the following ways,

1. As a buffer zone for incoming raw event data.
   This buffer zone can hold data for many months and enables data flows to be replayed.
   It delivers an economic data retention solution.
2. As part of an ETL solution enabling deferred data delivery as required.
   In addition to the buffer zone facility, data can be delivered in any format required by a particular application.
3. As an extension of business applications for the storage of large volumes of original raw records.

It is usually expensive to store granular data for long periods. The Bdsafe system provides this facility cheaply and the data can be accessed directly by applications.

4. As an MDM component providing a 'single point of truth' for incoming raw data.
   All applications requiring granular data can receive it from the same source. Data can be delivered as real-time data or as historic data.

3. System Architecture

The architectural approach taken is one of frugal or gandhian engineering. This is generally defined as "*the science of breaking up complex engineering processes/products into basic components and then rebuilding the product in the most economical manner possible*".

The simplest approach to take is simply collect files as they arrive and create external indexes. This is the approach taken in the Bdsafe system, with minimal extra steps to achieve stable performace.

For the Bdsafe system, the requirements were the ability to handle,

1. WORM data.
2. Event records.
3. Large volumes.
4. Fast storage.
5. Fast retrieval.
6. Low cost.

Event records are defined as data units made up of fields which record the occurrence of an event and which will not change.
Examples are,

1. Telco call records.
2. Log file records,
3. IPDR records,
4. Messaging records,
5. Financial transactions.

The data store partitions data by day.
In the multi-server version, each daily partition can be spread across multiple servers.
Data files are stored in a virtual file system.
Data files are typically stored in delimited text format, but original formats can be stored, and compressed formats can be stored.
The system uses a form of inverted file index.
Although indexes are applied at the record level, the original files can be extracted from the data store easily as the file is the basic data unit.

How does this stand in comparison to other approaches for storing large daily volumes of data?

3.1 Alternate Architectures

You could say that there are 3 main alternate approaches

1. RDBMS systems.
2. Map/Reduce systems.
3. NOSQL systems

3.1.1 RDBMS

RDBMS systems are good for

1. Transactional processing
2. Updating records
3. Complex predefined reporting and analytics.

They are very good solutions for these purposes, but slow and expensive for storing large daily volumes of data.

3.1.2 Map/Reduce

Map/Reduce systems are good for

1. Storing large volumes of data
2. Ad-hoc reporting

They are very good for experimental analytics, but slow and complex.

3.1.3 NoSQL

There are many systems of this type covering a wide range of approaches.
Some examples are
Hadoop (a wide column store),
Mongodb (a document store),
Berkeley DB (a key/value store),
but they are almost too numerous to mention. (See http://www.nosql-database.org/)
The pros and cons of these systems are really specific to each system. The fact that there are so many of them is an indication that requirements in the real world are very varied so it is important to match tools to requirements.

In our experience with storing and retrieving large data volumes, we have found that none of these alternate approaches are satisfactory in terms of price and/or performance.
Up to 200 million records a day with minimal index requirements, many systems will cope, but over say 500 million records a day or into the billions, with say 8 indexes or more, most systems either struggle or require significant hardware resources to perform the task.

The Bdsafe approach can be described in very simple terms as storing files in their original form and adding external indexes to enable flexible retrieval. It achieves high performance by not attempting to deliver complex analytics, there are many excellent systems that focus on analytics.

This approach obviously does not complete with the other approaches. On the contrary, it complements them and gives extra options to the end user.

For example, the same data set can be sent to several different analytic engines in parallel so that a set of results can be delivered which is better than the results of any single analytic engine.

3.2 Is there anthing unique about the Bdsafe system?

We believe the answer is yes.

It provides data retention (for long periods) without sacrificing data retrieval options, at very low cost.

It provides a single version of high quality data, usable by any other system, as and when required.

It can retrieve historic data at the record and the file level.

By using filter processes, it can retrieve (or more accurately, forward) current data as it is received.

It puts the user firmly in control of the data.

The domain of storing regularly large volumes of event data is one aspect of data management that still has no solid solution.
The Bdsafe system follows one direction for a practical solution.

4. Configurations

The Bdsafe data store is built on a single server core component.

Multiple servers can be used together to build a system of any size.

There is a java based control system in the full with a GUI which can be used to manage and query a multiple server system.

There is a stand alone CLI control process which can be run from any machine to manage and query a multiple server system.

There is a CLI process which can be run on the data store system itself.

The system does not require any particular hardware configuration. It can run on a single notepad or a server farm with hundreds of servers.

The main distribution is primarily for the RedHat Linux O/S but other systems have been used, for example Solaris.

4.1 Flexibility

The system design caters for the following options,

1. Run the core system on a single server.
2. Run multiple instances of the core system on multiple servers as a single unit.
3. Run different parts of the core system on multiple servers to address situation specific bottleneck points.
4. Easily export the data in original form to other systems to make the system easily replaceable, ie no tie-in.
5. User defined reliability levels.

The system can be used as a permanent or a temporary system.

An example of a permanent system is to provide data retention with easy data retrievability.

An example of a temporary system is to support a development project. Let's say a call-record management system is being developed. In this case, real call records can be fed to a Bdsafe data store with minimal disruption to the live systems. This data store can then replay any live data set (for say a rolling, previous 6 month window) to development systems at any time. Once the project is complete, the Bdsafe data store can be switched off. The cost of this temporary system for say 12 months could be something like 7k$ for a server and 5k$ for official product support if required.
Note that this could be 0k$ if an existing server is used and 0k$ for official product support if none is required.

Operational reliability levels can be matched to user requirements.
Raid-5 discs would be adequate for many cases where a few hours unavailability can be accepted.
Raid-1 discs would be adequate for more cases.
Multiple copies of an installed system can cater for all cases.

5. Basic Performance

An example installation using 3 commodity servers stores approximately 3 billion records in about 3 hours creating indexes on 7 fields. Due to linear scalability stemming from a 'shared-nothing' architecture, 10 severs in the same situation would handle about 10 billion records in 3 hours.

A query typically returns a record set of 1000 records within seconds and a record set of 1 million records in less than 1 minute.

6. Evolution.

6.1 Original Feature Set.

The original version of BDsafe had a fairly simple data presentation capability. It essentially serviced queries of the form,

“Select * where field = value and date-range = value”.

and used a proprietary query API.

The retrieved data sets could be fed directly into presentation systems, as CSV files. Early presentation systems were,

1. Analysts Notebook for Peacock and Time-line charts,
2. Google Earth for geovisualisations,
3. Spreadsheets for raw data presentation.

For complex analysis, the data (or data aggregates) were fed into appropriate analysis engines. For example link tables could be managed by feeding data into an RDBMS system, customer billing could be done by feeding the data into a billing system.

6.2 Features which have been provided to suit particular cases.

Over time, the following features have been delivered in particular cases,

1. The system can be used as backup system from which files can be retrieved.
2. The system can be used as backup system from which record sets can be retrieved.
3. The system can be used as a storage engine with a MySQL client front-end. This also delivers a JDBC capability, an ODBC capability, and a programming API for many programming languages.
4. A similar capability is in development with the PostgreSQL client front-end.

5. Aggregates can be created in parallel with the indexing phase and forwarded to other systems.
6. Rules based alarms can be triggered in parallel with the indexing phase.
7. Data compression can be invoked on stored data at any time.
8. Result sets can be formatted prior to delivery to suit the requesting application.
9. Multi-condition queries.

7. Free Trial system

A subset of the original system has been packaged to provide a general solution to managing large data sets.

It is offered as a free system.
The purpose of this is for us to obtain feedback from a more general user base. At the same time, it presents users with an opportunity to gain experience in the rapidly evolving area of large volume data management.

Official support for the system is available, but this is at a low cost.
The license to use the free system has no cost and standard support is from 5k US$ per year.
The maximum standard support cost for unlimited data on an unlimited number of servers (for a single system) is 25k US$.

The free system is a single server system with unlimited data storage. The limits in practice will depend on the number of discs attached to the server.

For example, for a trial, take 1 server with 6 x 2Tb discs, set up as raid 5 giving 10Tb of useable space. Assume 200 byte records with 7 fields indexed. Assuming data compression is used, this server would store something like 100Billion records.

Standard servers have loaded this volume of data at rates of 300 million records per hour per server.

8. Real world costing example (as at July 2012)

Consider a scenario where 1 billion call records are generated per day as per the above example.

1 commodity rack server could store and index (7 fields) 1 billion records in 3 hours.
With 6 x 2Tb discs (raid 5) giving 10Tb of space, the server could store 40 days of data.

If this server is used to store 1 month of data, 12 servers could store 12 months of data and in the 13th month, the oldest month can be recycled to be used for the current month.

Assuming a cost of US$7.5k per server, and optional support (see Note 1 below)

Cost to store 1 month, ie 30billion records is, for 1 server
1 x US$7.5k (+ optional US$5k) = 12.5k US$.
Cost to store 12 months, ie 365billion records is, for 12 servers
12 x US$7.5k (+ optional US$25k) = 115k US$.

If the data is stored in compressed form, 1 server could store 120 days, so,

Cost to store 3 months, ie 90billion records is, for 1 server
1 x US$7.5k (+ optional US$5k) = 12.5k US$.
Cost to store 12 months, ie 365billion records is, for 4 servers
4 x US$7.5k (+ optional US$20k) = 50k US$.
Cost to store 36 months, ie 1095billion records is for 12 servers
12 x US$90k (+ optional US$25k) = 115k US$.

Although DT2 does not measure things in this way, there is a current trend to quote price/Tb figures. As a loose comparison, the last case quoted above gives approx 500 US$/Tb of user data or approx 10 million records/ 1 US$. These figures include the cost of the servers.

Note 1. Support is optional and is US$5k per server per year with a cap at US$25k, ie support for the 6th server and above is free.

Note 2. These support fees are standard, irrespective of number users, number of CPUs per server, and irrespective of number of replicas of a system to achieve the desired reliability levels.