

Service Oriented Bulk Processing (SOBP)

November 18

2007

The concept of Service oriented Architecture (SOA), although popular in the OLTP world, is still foreign to the world of bulk data processing (OLAP/data warehousing and data mining). This isn't due to a lack of need; in fact, lack of service orientation is costing business tremendously specially since IT is facing data explosion – on one hand, cost of warehousing data is expensive, on the other, lost opportunity cost of not doing so might be even more expensive. This paper introduces the concept of service orientation to the world of bulk data processing.

SOA in the world of bulk data processing

Contents

What's new about Service Oriented Architecture (SOA)?	3
Bulk data processing – SOAP challenges	3
Challenges with the current bulk processing toolset	5
Expensive licensing	5
Problem with no license solutions (Lack of Enterprise capabilities)	5
Development	5
Execution and Scalability	5
Maintenance	5
Technology shortcomings	5
Proprietary nature	5
Inflexibility	6
Lack of Service Framework	6
Introducing SOBP – Service Oriented Bulk Processing	6
Ability to implement design patterns in the ETL world	6
Ability to automatically layer core set of enterprise services over ETL	7
Seamless integration with SOA world	7
Ability to exchange bulk data over the internet with minimal overhead (not XML)	8
Introducing dwSavvy	8
Case Studies	8
Data processing at a large bank	8
On Demand Provider of Business Intelligence	9
What does a typical small to medium sized project with dwSavvy look like?	10
Conclusion	10

What's new about Service Oriented Architecture (SOA)?

The concept of modular programming, building functions and fostering reusability, has always existed in the world of computing right from the early mainframe days. So what's new?

While these concepts are universal and have always been around, every new incarnation of the same concept comes with incremental advantages and better implementation. There has been a constant evolution in the way service providers (function) and service consumer (function call) handshake, and the logistics. Success of every such incarnation primarily depends on the ease, accessibility, efficiency, accuracy and reliability of the handshake. SOA is the latest incarnation. Computing needs today are a perfect storm for SOA:

- Web has provided a solid logistical framework for connecting people and processes globally
- Globalization has fostered rampant trade without boundaries
- Enterprise applications are at the forefront of driving business, leading to tremendous data processing needs
- Data explosion
- Open systems and inexpensive commodity hardware

A sum total of what's needed to enable efficient business growth and exploit technological advancements is SOA. It enables distributed computing and promotes standardization and rationalization of business rules, as "services", across the enterprise and eventually across the globe. With SOA, what started in the programming world as sharing of small code as functions at the elementary level (for instance a mathematical function), has lead to sophisticated services that encapsulate complete business function like submitting and processing a loan or a credit card application. The old world of functions was more atomic in nature and logistically co-located. With SOA, these have evolved into sophisticated full service business functions that spread out globally but are easily accessible over the web irrespective of distance and platform differences.

Central to enabling SOA is "web services". Although the concept has been around ever since the maturity of web, it started delivering on its promise only with the advent of SOAP (Simple Object Access Protocol). Key to the success of SOAP is its use of XML as the communication vehicle that can easily be understood by any platform. The complexity of distributed applications comes from various platforms using proprietary metadata formats. The only way they can communicate is by understanding each other's metadata definition language. By using XML, i.e. wrapping data within its metadata in clear text, SOAP resolves this issue quite effectively.

Bulk data processing – SOAP challenges

XML, the very aspect of SOAP that enables SOA is detrimental to bulk data exchange/processing. XML's use of metadata tags around every single data element caused data explosion. For large data sets this would mean significant overhead on all aspects of data handling -- extract, XML compose and decompose, serialize and deserialize, compress and decompress, transfer etc. XML could increase the

overall data size by as much as 10 times or more. This poses a significant overhead to processing very large data sets, so much so that with the current set of tools and techniques SOA is only a dream for the bulk data processing world. This need however is out there begging for a solution.

Today's popular tools for bulk data processing

Let's take a dive into tools and techniques currently available for bulk data processing:

Good old scripts

Traditional means for bulk data processing. Some of the popular ones are Perl, python, java, VB etc. These continue to be popular as they are easily accessible, well documented over the web, and fairly easy to code. This approach however is completely devoid of metadata management, which is a huge drawback. Each script that parses a file needs to have the file metadata information embedded within it. This metadata information that is mixed with business processing rules is not easily readable, shareable or accessible to the rest of the enterprise. In large organizations it's typical to have hundreds of files getting parsed and processed by thousands of scripts. Imagine the impact of changing a few files formats; for instance, addition of new columns which is a very common occurrence. In my experience, in a healthy enterprise, even the most stable file undergoes at least 3 to 4 changes within a year.

Database processing (SQL scripts and stored procedures)

As relational databases started fine tuning themselves for bulk data processing they pretty much became the default engine for fast and easy bulk data processing with wide acceptance – they still do to a large extent. They have an advantage over scripts as file metadata is centrally stored and managed by the database engine. Parsing the file format and loading it into a database table was now a onetime activity. Impact to change in file format can now be contained within one script/program that parsed the file and loaded it into the database. However, databases have a few problems of their own:

1. Most of the databases are quite expensive with the exception of open source databases.
2. Their proprietary procedural language locks you into a single platform.
3. Taking advantage of some of the essential parallelism techniques, for fast processing, like component, pipeline and process parallelism is either difficult or impossible.
4. Performing data transformations while users access the database is a problem both from a performance as well as data integrity perspective.
5. Lack of GUI based drag and drop style programming platform.

ETL

With the advent of GUI based tools to perform data processing like Data Stage, Informatica, AbInitio etc. bulk data processing got a face lift in the enterprise world. These technologies, for the most part, provide a way to code ETL graphically – complex scripting is still not visual though. They foster simple reusability but lack the infrastructure needed to develop “data services”.

Challenges with the current bulk processing toolset

Expensive licensing

Popular database and ETL/ELT technologies are all exorbitantly licensed per CPU core (dual core CPU is priced higher than single). For small and medium sized companies these costs could be prohibitive. Typically in the world of data mining and analytics some amount of data analysis is required before realizing the full potential of value hidden in data – license expenses come in the way of realizing such potential.

Problem with no license solutions (Lack of Enterprise capabilities)

To overcome the licensing issue, several companies choose to go with freely available scripting tools like Perl or Python or even Java. These languages, although extremely powerful, take the focus of building a data warehouse away from business logic to programming and managing the infrastructure in a non-enterprise environment. A whole slew of issues arise:

Development

Not easy to code; takes longer to code and debug. Typically, it's hard to follow standards and best practices. Very likely every programmer will follow a different technique towards the same end.

Execution and Scalability

1. A typical data processing environment (data warehouse) will have at least a few 100 scripts that are dependent on each other and are also time bound. Managing this environment quickly gets out of hand without the use of an appropriate scheduling and workflow tool.
2. For faster processing, one would typically run as many scripts in parallel as possible which is an added dimension to the above problem.
3. By now, almost everyone in IT has probably experienced data explosion – especially in a healthy company. The most cost effective model to combat scalability is to cluster generic HW in a grid. Getting a grid going using scripts is a project on its own.

Maintenance

Maintaining these scripts without the original programmers is expensive and difficult. These scripting languages don't naturally lend themselves to proper documentation.

Technology shortcomings

Proprietary nature

Each ETL technology has its own scripting language and set of idiosyncrasies that can only be mastered after gaining substantial experience with the tool set. This gives rise to unnecessary learning curve – reading a table/file, sorting it and joining it with another table/file is universal across all technologies, but they all have a different way of coding it and performance tuning it. Same is true with applying basic programming constructs like conditions, loops etc. – mastering syntax should not be the focus a programming exercise, it takes valuable time and energy away from implementing business rules and being creative in general.

Inflexibility

Every ETL technology comes with a set of “out of the box” features, which are well documented and easy to use. However, the moment you outgrow the vanilla feature set, it gets very difficult to achieve your ends. There’s no visibility or access to “inside the box”.

Lack of Service Framework

None of the ETL technologies in the market today provide a way to implement design patterns for bulk data processing (data warehousing). Design patterns are not a new concept, they are quite prevalent in the application (java, C++) world but slow to catch on in the ETL world. Most ETL technologies provide a means of reusability as long as the metadata is locked – not otherwise. Take for instance a simple ETL problem of loading a file into a database table with some data quality checks. It’s easy enough to write a mapping to load this data. What if your shop needs to load 50 to 100 files? As long as the file format remains the same these mappings can be parameterized and reused. If however the file formats differ, one would have to write one mapping for each format. If all the 50 files are of different format, one would have to write 50 mappings. This is the basic essence behind design patterns – the fact that a file needs to be loaded with some data quality checks did not change, so why should the program change. The reason today’s tools fail to achieve this flexibility is because the metadata for various objects (files/tables) is burnt into the code at design time as oppose to run time.

Introducing SOBP – Service Oriented Bulk Processing

The basic idea behind SOBP is to enable the concept of **services** in the world of bulk data processing (data warehousing). What would a service enabled platform for bulk data processing entail?

1. Ability to implement design patterns in the ETL/ELT world
2. Ability to automatically layer core set of enterprise services over ETL/ELT
3. Seamless integration with SOA world
4. Ability to exchange bulk data over the internet with minimal overhead (not XML)

Ability to implement design patterns in the ETL world

In any typical data warehousing environment 70 to 80% of the requirements are standard and common across most installations. A small % is dependent on data volumes – very large vs. large vs. not large – and a very small % is dependent on specific industry or installation. So why do we reinvent the wheel? Other than data warehousing concepts being standard – thanks to Bill Inmon and Ralph Kimball – almost everything else is engineered afresh. In fact, even within the same organization, given the current ETL toolsets, the code base is replicated with minor modifications. Following are examples some common patterns:

1. File loads into the data warehouse
 - a. Load any file into any given table (option to create table if nonexistent, using a predefined format and naming pattern for tables and columns)
 - b. Detect duplicate load and abort; send email alert
 - c. Perform data quality checks

2. Refresh target table given a source
 - a. Refresh a target table using either a file or a table as source
 - b. Automatically detect incremental vs. full refresh and process accordingly
 - c. Perform data quality checks
3. Refresh dimension
 - a. Refresh dimension using either a file or table as source
 - b. Automatically perform type1, 2 and 3 changes for preconfigured columns/attributes
 - c. Allow a column to go from type 1 to 2 or vice versa using configuration parameters
 - d. Keep track of total type 1 and type 2 changes for each load
 - e. Perform data quality checks

Unfortunately, ETL/ELT technologies available in the market today don't provide an ability to implement generic patterns. This leads to developers cutting and pasting the same code over and over again. Typically for a large organization, worst case scenarios play out – every developer finds a different way of implementing the exact same pattern, resulting in lack of uniformity which in turn leads to data quality issues and increased support and maintenance cost.

Ability to automatically layer core set of enterprise services over ETL

If you are in the market for buying a vehicle you'll notice that more of the features considered luxury in recent past are becoming standard – my favorite is the side impact airbags and rollover stability control in SUV's; safety shouldn't be an option. My point being, when we talk about software as a service some required features shouldn't be optional – core services. In fact, there shouldn't be a way around it – try selling a car without seat belts. Here's a list of services that I would consider core for bulk processing:

1. Scalability – Clustered solution over commodity hardware
2. Parallel processing
3. Metadata Service
4. Fault Tolerance
5. Scheduling
6. Process Monitoring
7. Process Control
8. Trending of runtime facts like job run time, start time, end time etc.
9. Batch framework
10. Data Profiling
11. Data Quality
12. Archiving
13. Logging

Seamless integration with SOA world

Key services like metadata service, process-control, data transfer etc should be exposed as web services – SOAP. This'll enable web services in the application world to seamlessly integrate with the bulk processing world. Today, this interface (hand-shake) is only via files and tables.

Ability to exchange bulk data over the internet with minimal overhead (not XML)

XML's ability to wrap metadata around data – including relationship between data element – is what makes it a powerful tool for data exchange across disparate platforms. Unfortunately, this very strength makes it unusable for bulk data processing – meta tags around every single data element increase the original data set size by several folds (10x or more).

Following are the dimensions around which this problem can be overcome:

1. Ability to exchange massive amounts of data at high speed (compression enabled)
2. Ability to exchange metadata
3. Ability to be platform independent
4. Ability to transfer data securely

Introducing dwSavvy

dwSavvy is the first platform, built ground up, for Service Oriented Bulk Processing – SOBP. Here's how it excels in service orientation:

1. Ability to convert any identified ETL/ELT pattern into a data service. These data services are independent of file/tables formats.
2. Ability to wrap core services over any ETL/ELT
3. All the data and metadata services are exposed as web services – seamless integration with SOA
4. High speed bulk data exchange enabled over the internet via a unique combination of web services (interface) and established file transfer protocols like ftp, sftp, scp etc.

For more information go to www.dwsavvy.com

Case Studies

Data processing at a large bank

A large bank maintains a huge data warehouse, processing several terabytes of data each month. Like the rest of the industry, it's experiencing an explosive growth in terms of:

1. New data processing need
2. Cost of project execution
3. Cost of ongoing maintenance

To overcome cost and speed to market issues they outsourced their data processing to a premium IT outsourcing vendor. Although the strategy had an immediate impact to the bottom line, in time though, it was clear that more needed to be done to keep up with the project speed and quality demands without exponentially increasing cost.

With dwSavvy, the bank was able to both increase speed to market and reduce cost (development, testing and maintenance) by 50 to 70%. Here's how:

1. Identification and implementation of ETL patterns: 70 to 80% of ETL being executed followed the same set of 7 to 10 patterns. Right from simple patterns like loading a file into a table with health checks to complex ones like performing type 1 and type 2 changes on a slowly changing dimension. After the implementation of these patterns, with dwSavvy, executing a new project was a matter of applying the same ETL patterns to new source objects (files/tables). With dwSavvy an ETL pattern gets implemented as "data service". Once a data service is attached to the appropriate objects (files/tables), like a parameter, it becomes an executable – job. Attaching a data service to an object is done using simple 100% web based GUI.
2. Production support: Every piece of ETL runs within the framework of core services (see "Ability to automatically layer core set of enterprise services over ETL") thus minimizing production support cost -- issues like duplicate run, not following dependencies etc are avoided.
3. Metadata changes: dwSavvy shields existing ETL from source format changes – the only change that's required is in the metadata repository to register new formats.
4. Source format agnostic: There was a need to ingest files with the exact same content, from several different sources, but different formats. With dwSavvy this problem was elegantly resolved with a single data service being attached to all the different file formats.
5. Out of the box data services, packaged with dwSavvy, were immediately usable with little or no customization.

On Demand Provider of Business Intelligence

An on-demand provider of data warehouse/business Intelligence services is looking for an ETL infrastructure enabler that can maximize its profits while providing significant cost benefits to its customers. After careful consideration and comparing existing technologies available in the market place they zero in on dwSavvy as their platform of choice. Following were some of the considerations:

1. Commodity hardware that can scale horizontally in a cluster – Inexpensive processing power.
2. Inexpensive software license – Most of the ETL and ELT (database) technologies, other than open source, cost close to \$15K to 20K per CPU core plus 20% support. Typically, this is just the base price – doesn't include goodies like data profiling, data quality etc.
3. Integrated Core services: Essential services like clustering (horizontal scalability – linear), fault tolerance, scheduling, process control, central monitoring, data profiling, data quality, batch framework, batch/job recoverability from failure, logging etc. should be layered seamlessly over any customer ETL.
4. Reduce Labor cost – Optimizing on labor is a function of the following:
 - a. Extreme reusability: Identify ETL patterns and implement data services. This significantly reduces the code base resulting in reduced labor cost, increased speed to market, reduced on-going maintenance, reduced bugs, excellent code uniformity and excellent data quality.
 - b. Reduce maintenance cost: Significantly reduced code base helps reduce the cost of testing and maintenance. Metadata changes do not impact ETL as long as the ETL pattern does not

- change – all ETL code is created dynamically, based on the implemented ETL patterns, at the time of execution. Changing an ETL pattern or augmenting existing one is simple and painless – all the existing processes implementing the pattern automatically inherit the change. This completely eliminates code changes and significantly reduces the cost of testing.
- c. New projects are simply a matter of snapping in the right ETL patterns around custom business logic.
 5. Multi-tenancy – built-in capability to handle data from multiple clients.
 6. Web Based Interface: A web based user interface to manage the entire backend infrastructure. Nodes can be added and removed from the cluster in the middle of batch processing, without impacting the whole batch.
 7. Ability to seamlessly integrate the backend ETL environment with customer's SOA framework.

What does a typical small to medium sized project with dwSavvy look like?

1. **Engagement** (2 to 3 weeks, 1 to 2 resources): dwSavvy ETL architects understand your pain points and assess your ETL environment. Deliverable – Environment assessment document.
2. **Identify data services** i.e. ETL patterns (2 to 3 weeks, 1 to 2 resources): dwSavvy ETL architects identify data services to be implemented. They perform a quick gap analysis between existing, out of box, data services and what's needed. Deliverable – Project plan with time and cost to (1) use identified data services, (2) customize existing data services and (3) build new ones. **Note:** As the repository of data services increase, the need to build new ones shall reduce significantly.
3. **Implement and test data services** (4 to 5 weeks, 1 to 2 resources): dwSavvy data service programmers.
4. **Understand and code business logic** (not driven by any pattern): This step can be executed by any ETL programmer since it's no different from current ETL technologies. ETL programmers can use any ETL technology; we recommend the use of open source ETL tool from Clover (www.cloveretl.net) for best results.
5. **Perform end to end testing.**
6. **Setup and configure production** environment (2 to 3 weeks, 1 to 2 resources): Enter cluster, server, object (files/tables), data-service and job metadata. Schedule and define batch and job dependencies.

Note: Time and resources outlined above are for a typical small to medium sized project. These may differ significantly based on the size and complexity of the environment.

Conclusion

The benefits of service orientation in the world of application development are clearly evident. There's an urgent need for the same within the world of bulk data processing, especially since organizations today are experiencing a tremendous growth in data that's begging to be warehoused and mined

efficiently. There's tremendous cost savings in sharing data services across the enterprise – proven in the applications world (SOA). It also brings organizations closer to unifying and centralizing data processing rules – single version of truth, both in terms of data and metadata.

dwSavvy is the first product in the market to introduce the concept of Service Oriented Bulk Processing (SOBP) and implement it successfully -- next logical step in the evolution of ETL/ELT technologies. Central to its success is the concept of “Core Services” and “Data Services”. Core services are essential services like clustering, fault tolerance, batch framework, data quality, error handling etc. that are automatically wrapped around every ETL that runs on this platform. Data Services, on the other hand, implement ETL patterns that are extremely flexible and reusable; they overcome the traditional file/table format/layout boundaries – for instance, file layout changes do not impact dependent ETL.

Bottom-line: Significant cost savings (50 to 70%) on build and maintenance of all data warehousing and bulk data processing needs.