Understanding Business Trends from Data Evolution with Tornado

Azhar Ait Ouassarah, Nicolas Averseng, Xavier Fournet, Jean-Marc Petit, Romain Revol, Vasile-Marian Scuturici

Understanding Business Trends from Data Evolution with Tornado

Azhar Ait Ouassarah* †, Nicolas Averseng†, Xavier Fournet†,
Jean-Marc Petit*, Romain Revol † and Vasile-Marian Scuturici *
*Université de Lyon, CNRS, INSA-Lyon, LIRIS, UMR5205
20 avenue Albert Einstein, F-69621 Villeurbanne Cedex, France
†Systar
171 Rue Bureaux de la Colline, 92210 St Cloud, France

***Abstract*—Nowadays, every company could understand how its business evolves from the data (deluge) generated by its activities. Roughly speaking, two types of data co-exist: historical data and real-time data from which business analysts have to take their decisions in a timely fashion. In this context, the notions of time (application time and transaction time) and traceability turn out to play a crucial role to understand what happened in the company and what is currently happening. Tornado offers a full-fledged platform to deal with such data and is based on two key features: 1) a bi-temporal DB specifically designed for handling historical and real-time data, 2) a GUI that aims to facilitate query formulation for business analysts. In this demonstration, we provide the key resources to let the visitors play with the Tornado functionalities to interact with predefined data.**

I. Introduction

Companies are operating in very dynamic and complex environments that require from their managers agility and ability to make proactive operational decisions, in order to maintain or improve their business. Traditionally managers rely on Business Activity Monitoring (BAM) [1] to take operational decisions. BAM aims to provide real-time access to critical business performance indicators. Thus managers can have a deep insight of what is currently happening in their business and then take rapid and effective decisions. BAM gathers its information in real-time by analysing data streams from multiple sources. BAM solutions often rely on Complex Event Processing (CEP) [2] can also detect interesting patterns of events, e.g if events A and B happen then C happens too. Nevertheless these technologies are limited because they only focus on real time information rather than using existing historical data. Exploiting historical data is covered by the Business Intelligence (BI) domain [3] which provides access to past performance indicators by analysing information stored in data warehouses. It enables users to understand what happened in the past and help them to prevent the mistakes in the future. Tornado, a platform developped by Systar (http://www.systar.fr/), addresses analyst's needs to cover BI and real-time monitoring use cases. Even if some contributions exist in the Data Stream Management to deal with real-time and historical streams [4], [5], Tornado is, to the best of our knowledge, the first comprehensive data-intensive decision support system allowing to exploit real-time data streams and historical data in a transparent manner.

Tornado has been designed to be easily deployable. Thus it do does not require specific hardware equipment and can be deployed on commodity hardware at a low cost ownership. Creating or updating applications does not require deep technical knowledge. Tornado embeds a proprietary NoSQL bi-temporal and column-oriented database management system (2TDBMS) that is continuously fed with data by external data sources. In order to cope with real-time constraints the query processing engine uses different types of continuous queries. Analysts with limited technical skills can create new analyses, tailor their dashboards and see the results of their modifications immediately on production data. Thus these applications evolve as user's needs evolve to meet the business evolution.

As Tornado combines both real-time monitoring and BI functionalities, it enables analysts to consider operational data in the past, present, or future using the same tool. It provides consolidated views of traditional real-time analyses with complete temporal historical analyses. The query engine allows typical user scenarios as:

- Replay of past situations with exactly the same information as when they occurred live;
- Investigation for audit and traceability to provide:
  - In depth analysis of a situation that occurred in the past, which could include answers to questions such as what happened, when, why and where specific actions are taken;
  - On demand simulations of the evolution of past situations;
  - Parallel comparison of the evolution in time of two situations, such as a present time process and the behavior of the same process yesterday;
- Risk assessment evaluations which may be:
  - Based on past events and data;
  - Based on forecasted events and data.

Tornado has to face two main performance issues: processing high volume of data generated by business activities and guaranteeing fast response time for users monitoring business activities through a GUI interaction.

In this demonstration we will highlight the two main features of Tornado: 1) An unified GUI to query and visualize

both real-time and historical data. 2) A proprietary NoSQL, Bitemporal and column-oriented DBMS (2TDBMS) that is specifically designed to handle both historical and real-time data.

II. Tornado Description

A. Architecture

Tornado is based on a service oriented, event-driven implementation using Java™ language. It is structured into three loosely coupled functional layers: Absorption Layer, Logic Layer and User Interface Layer. *Absorption Layer*: Tornado uses unobtrusive, agentless technology to collect, process and analyse real-time data as well as historical data. It is based on the Apache Camel engine that offers a wide range of possibilities to pull data from various types of sources.
Logic Layer: The logic layer embeds a 2TDBMS specifically designed and optimized for Tornado. The power of that DBMS is its ability to handle both real-time streams and historical data and provide an unique interface to access to them.
User interface layer: The user interface layer allows any web browsers using Adobe Flash technology to display information and from the logic layer. This layer is fully integrated with the logic layer interface to offer seamless and rich interaction with the analysts with visual data manipulation, navigation, as well as analyses (Fig. 1) based on its needs.

B. Bi-temporal and Column-oriented Features

The underlying DBMS is bitemporal, i.e it supports two orthogonal time dimensions: the *transaction time* [6], or *system time*, (tt) and the *valid time* [7], or *application time*. The support of transaction time enables to maintain a history of all modifications that occurred in the DB. The support of the valid time enables to maintain the history of data as it evolves in the modeled reality.

The DBMS has the particularity of being column-oriented [8], [9] which means that each attribute is stored in a separate column and its values are stored successively on disk. This contrasts with the row-oriented DBMS where attributes of the same tuples are stored consecutively. There are two reasons behind choosing a column-oriented database management system rather than a row oriented one. First, column stores outperform row oriented database system on analytical workloads such as those found in business intelligence and decision support application [10]. Secondly, the column oriented approach enables creation/suppression of columns with limited performance impacts, simplifying the data model update.
Tornado stores data in an append-only mode. Each attribute is timestamped with two temporal attributes: 1) a valid time interval, provided by the application, that indicates the time during which the attribute's values are valid, 2) a transaction time, implicitly provided by the system during the transaction commit, that indicates when attribute's values are inserted into the DB.

The query engine integrated in the logical layer is optimized for three classes of temporal queries:

1) Time Travel: Time travel is a feature that enables to build a view of a subpart of the database at a particular state considering a valid time and a transaction time instant. It consists of all valid and accessible data at that state. As an example, a user might be interested in all unpaid orders in September 2nd 2014 midnight as database is currently recorded.

2) Temporal Aggregation: Tornado enables to compute aggregation functions over valid time while the transaction time is fixed to an instant. We consider two cases: aggregations over an instant and over a temporal range. In the first case, termed also *Instantaneous Aggregation* [11], we aggregate all valid tuples at an instant, e.g what is the number of payments with status='in process' in September 2nd 2014? In the second case, we aggregate all tuples associated to a time range using a temporal predicate. If we consider Kaufmann and al's classification of time ranges in temporal aggregations in [12] which uses the window concepts from data streams systems [13], Tornado supports three types of time ranges : 1) Tumbling Window, i.e aggregation is performed on non overlapping intervals (e.g the number of new payments per day). 2) Sliding Window (e.g the number of new payments every day for the last 10 days), 3) Landmark Window, i.e the intervals share the same interval begin (e.g the number of new payments from the beginning of the month up to each day).

3) Temporal Join: Tornado implements temporal join that simply extends the classical join with an additional predicate over the valid time intervals. Thus two tuples match if they satisfy two conditions: 1) the predicate over the non temporal attributes is satisfied and 2) the valid time intervals satisfy the temporal predicate. Tornado proposes 7 predicates like *A intersects B*, *A beging during B*, ...

C. Temporal Query Engine

Tornado uses a *dashboard* as an user interface primitive allowing the analysts to visualize analyses and to navigate in time to understand how they evolve. A dashboard is composed of one or several graphical elements (diagrams, charts, datagrids, ...) termed *pagelets*. Each graphical element displays data returned by underlying queries. The use of an interactive GUI requires short response time from the database system to avoid any unpleasant display lags. In order to display a complete dashboard, the update time of associated queries must respect these performance constraints, despite their complexity and the growth of the DB size. In our case, all queries are completely specified once dashboards are created or modified. Our strategy consists in computing all defined analyses as data is collected by Tornado and storing results in the DB for a future use. Thus in Tornado an analytical query, e.g as TPC-H ones, is executed as:

- several simple persisted *continuous queries* [14].
- one more elaborated *on-demand query* to provide the answer to the pagelet.

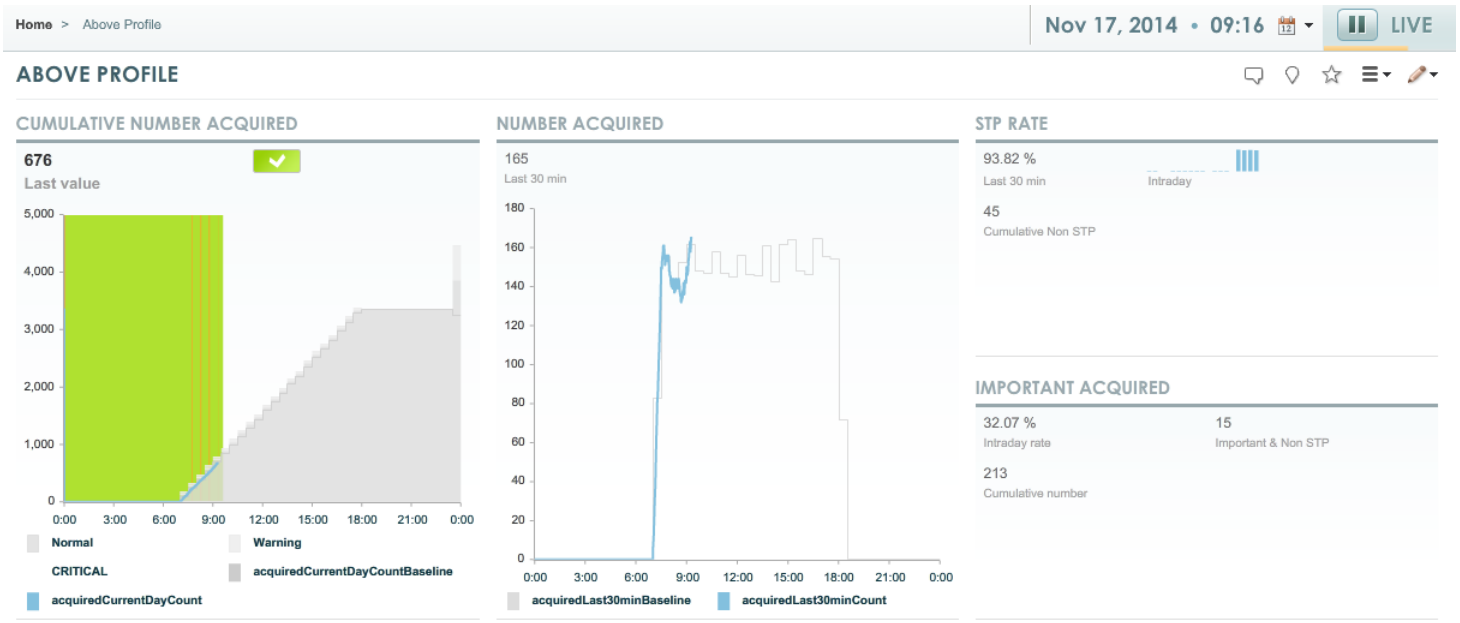

Fig. 1. Screenshot of a dashboard

1) Persisted Continuous Queries: For each analysis, one query is implemented and is binded to a *rhythm*. We define a *rhythm* as a partition of the valid time domain into contiguous and equal-length time intervals, and a new attribute is added to the data model. For each interval of that rhythm Tornado executes the query on data associated to it considering their most recent state according to the *transaction time*. The result is valid on the computation interval and is stored in its dedicated attribute in the database for future uses. This type of queries are continuously executed by Tornado's query engine as data is available, i.e either collected by the system from external sources or produced by another analyses.

2) On-demand Queries: This type of queries is executed as long as it is necessary to refresh the pagelet it is bind to. The refresh can either be automatic or caused by the analyst when he interacts with the GUI. The *on-demand queries* perform time travels and temporal joins against the DB that contain both collected data and the results of the continuous queries.

III. Demonstration

For our demonstration we use an application inspired by Systar's real-life workloads that was specifically implemented for demonstration and test purposes. It corresponds to a fictive organization in charge of supervising payment transactions between companies (Fig. 2). This application contains a set of dashboards representative for the platform capabilities. Besides it embeds a configurable data generator used to evaluate Tornado's behaviours to handle large data sizes. We use two parameters to set the size of generated data: *vt instant start* of the simulation and *number payments/minute*. The simulator works in two successive phases. First it injects into the platform historical data from *vt instant start* to the present time. Then the generator injects real-time data from *vt instant start*. At the rate of 250 payments/minute, the total size of data stored in the DB is 1.7GB/day: 0.2GB/day is the data collected by Tornado and the rest are the results of the *persisted continuous queries*

We assume that the application has reached the real-time step at the beginning of the the demonstration. The visitor of the demonstration will have the opportunity to understand Tornado through predefined queries. Then she will be invited to design its own pagelet and then explore analyses.

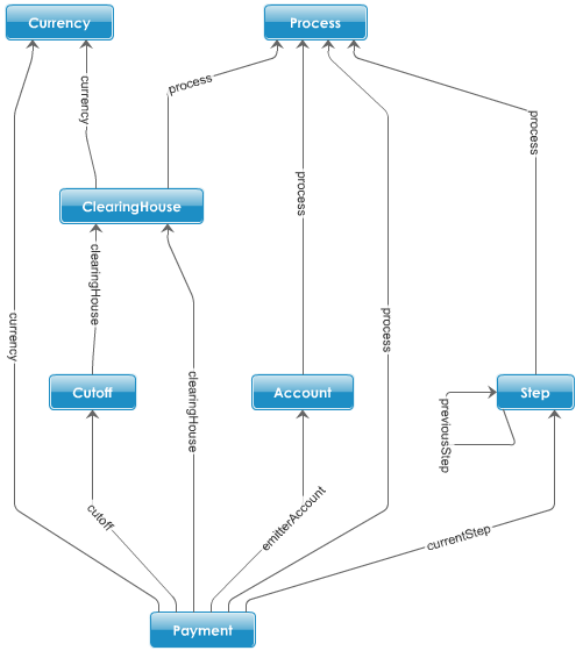

Fig. 2. Graphical data model of the scenario

A. Exploring Analyses

In the first part of the demonstration, the visitor will use the dashboards to monitor the company's activity and experiments the platform capabilities. By default, the dashboards display the real-time values of analyses. The visitor can display the historical values thanks to the *time machine* (Fig 3). It is a graphical feature that enables to navigate through time (according to both valid time and transaction time) and consider the dashboard at a certain time. Thus the visitor can for example replay past situations with the same conditions as they occured.

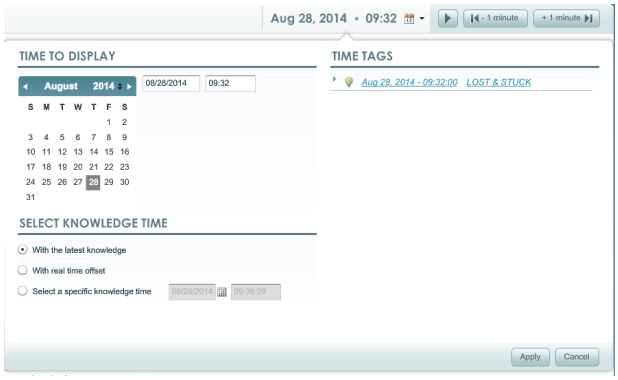

Fig. 3. Screenshot of the time machine

B. Designing a Pagelet

In the second part, we design a pagelet that displays, e.g *the number of new payments up to each day for the last 5 days computed every hour*. The design of a pagelet is in two steps: 1) implementing analyses 2) designing the pagelet's GUI.

1) Defining Analyses: We implement, using the GUI, all continuous queries that we will need for our pagelet. For each of them we specify the aggregation function to perform, the type of window to use, the rhythm... Considering the previous example, we define one continuous query that computes every hour the number of new payments since the beginning of the day.

2) Designing Graphics: In the second step, we define first the *on-demand query* in charge of returning results that are displayed on the pagelet. Then we also define the type of graphic to display these analyses.

IV. Conclusion

We have introduced Tornado, a platform embedding a column-store bi-temporal database system, devoted to historical and real-time bi-temporal data. Its main feature is to bring on manager's desktop precise and relevant indicators on its business. Tornado relies on two kind of technique: first, the data are summarized with relatively simple continuous queries and second, more elaborated queries providing key indicators to managers.

We plan to conduct experiments of Tornado with the TCP-BiH benchmark [15], which is a bi-temporal extension of the TPC-H benchmark. We are also working on multi-query optimization techniques, and also incremental pre-computation mechanism introduced in [16].

References

[1] D. W. McCoy, "Business activity monitoring: Calm before the storm," *Gartner Research*, 2002.

[2] D. Luckham, *The power of events*. Addison-Wesley Reading, 2002, vol. 204.

[3] H. J. Watson and B. H. Wixom, "The current state of business intelligence," *Computer*, vol. 40, no. 9, pp. 96–99, 2007.

[4] S. Chandrasekaran and M. Franklin, "Remembrance of streams past: Overload-sensitive management of archived streams," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 348–359.

[5] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, and J. M. Hellerstein, "Enabling real-time querying of live and historical stream data," in *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on*. IEEE, 2007, pp. 28–28.

[6] R. T. Snodgrass, M. H. Böhlen, C. S. Jensen, and A. Steiner, "Adding transaction time to sql/temporal," *ISO-ANSI SQL/Temporal Change Proposal, ANSI X3H2-96-152r ISO/IEC JTC1/SC21/WG3 DBL*, vol. 1101, p. 143, 1996.

[7] R. Snodgrass, M. Böhlen, C. Jensen, and A. Steiner, "Adding valid time to sql/temporal. ansi x3h2-96-151r1, iso–ansi sql/temporal change proposal, iso," IEC JTC1/SC21/WG3 DBL MCI-142, Tech. Rep., 1996.

[8] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil *et al.*, "C-store: a column-oriented dbms," in *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 553–564.

[9] P. A. Boncz, M. Zukowski, and N. Nes, "Monetdb/x100: Hyper-pipelining query execution." in *CIDR*, vol. 5, 2005, pp. 225–237.

[10] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. row-stores: how different are they really?" in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 967–980.

[11] C. Dyreson, F. Grandi, W. Käfer, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, D. Nonen, E. Peressi, B. Pernici, J. F. Roddick, N. L. Sarda, M. R. Scalas, A. Segev, R. T. Snodgrass, M. D. Soo, A. Tansel, P. Tiberio, and G. Wiederhold, "A consensus glossary of temporal database concepts," *SIGMOD Rec.*, vol. 23, no. 1, pp. 52–64, Mar. 1994. [Online]. Available: http://doi.acm.org/10.1145/181550.181560

[12] M. Kaufmann, P. Vagenas, P. M. Fischer, D. Kossmann, and F. Färber, "Comprehensive and interactive temporal query processing with sap hana," *Proceedings of the VLDB Endowment*, vol. 6, no. 12, pp. 1210–1213, 2013.

[13] L. Golab, S. Garg, and M. T. Özsu, "On indexing sliding windows over online data streams," in *Advances in Database Technology-EDBT 2004*. Springer, 2004, pp. 712–729.

[14] A. Arasu, S. Babu, and J. Widom, "An abstract semantics and concrete language for continuous queries over streams and relations," 2002.

[15] M. Kaufmann, P. M. Fischer, N. May, A. Tonder, and D. Kossmann, "Tpc-bih: A benchmark for bitemporal databases," 2013.

[16] M. Armbrust, E. Liang, T. Kraska, A. Fox, M. J. Franklin, and D. A. Patterson, "Generalized scale independence through incremental precomputation," in *Proceedings of the 2013 international conference on Management of data*. ACM, 2013, pp. 625–636.