A*DAX-DAT: A Toolkit Framework for Big Data Analytics

Htoo Htet Aung  
Institute for Infocomm Research,  
Singapore  
aunghh@i2r.a-star.edu.sg

Shili Xiang  
Institute for Infocomm Research,  
Singapore  
xiang@i2r.a-star.edu.sg

Paul Min Chim Lim  
Institute for Infocomm Research,  
Singapore  
limmc@i2r.a-star.edu.sg

Gim Guan Chua  
Institute for Infocomm Research,  
Singapore  
ggchua@i2r.a-star.edu.sg

Shuqiao Guo  
Institute for Infocomm Research,  
Singapore  
guosq@i2r.a-star.edu.sg

Ang Loon Chan  
Institute for Infocomm Research,  
Singapore  
chanal@i2r.a-star.edu.sg

Mun Thye Mak  
Institute for Infocomm Research,  
Singapore  
mmtmak@i2r.a-star.edu.sg

Wee Boon Koh  
Institute for Infocomm Research,  
Singapore  
kohwb@i2r.a-star.edu.sg

Wee Siong Ng  
Institute for Infocomm Research,  
Singapore  
wsng@i2r.a-star.edu.sg

Abstract — A*STAR Data Analytics Exchange ("A*DAX") is a scalable platform built upon open architecture and it is designed to facilitate the secure management and analytics of urban data. In this paper, we introduced A*DAX-DAT, which is the latest development of A*DAX. A*DAX-DAT (or DAT for short) is the analytics component of the A*DAX Platform. DAT aims to provide advanced analytics functionalities to A*DAX in such a way that: (i) it is extensible, (ii) it is easy to use for both end-users and developers and (iii) it is efficient in handling data regardless of variety, volume and velocity. DAT consists of three components: (a) a set of framework specifications for analytics libraries, (b) dynamic analytics libraries following the aforementioned framework specifications and (c) tools to develop/use the dynamic libraries either in standalone mode or in client/server mode. This paper presents the design of DAT components with implementation details before it discusses sample use-cases of DAT. Finally, it concludes with discussions on future works on DAT.

Keywords— data-analytics; analytics-toolkit; A*DAX; design, urban analytic; data platform

1. INTRODUCTION

As the urbanization process paces on, cities around the world become larger and larger in terms of both area and population. According to [1], about half (50%) of the world’s population is currently living in cities and, every week, people dwelling in cities is increased by 1.5 million. By 2020, it is estimated that there will be 2.2 billion urban dwellers – a 500 million growth to today’s 1.7 billion. This has led to an outburst of data size city administrations, traffic controllers and even commercial businesses collect, maintain and share.

The Agency for Science, Technology and Research (A*STAR) of Singapore launched the Urban Systems Initiative, a five year multi-program, in 2012 in order to help cities’ stakeholders with rapid urbanization challenges – among which is data collection, maintenance, sharing and analytics. It aims to enable the development of solutions for complex urban challenges in collaboration with the relevant government agencies and industries to enhance the competitiveness of Singapore in the construction of “smart-city.”

A*STAR Data Analytics and Exchange Platform (A*DAX) [2,7,8,9] is built in order to effectively and efficiently merge, store, access and share securely among programs in the urban initiative as well as with third-parties. It also serves another purpose of the urban initiative, which is to encourage companies, research institutions and individual developers to fuse data from different domains and develop new applications and/or services for better social good.

Since A*DAX is supposed to handle variety of data sources, it is designed to handle and store different formats of incoming data – including structure data, semi-structure data and unstructured data. A*DAX also include back-end modules – such as query manager to provide efficient data access, access control to enforce data ownership and sharing and data analytics to provide both basic and advanced analytics functionalities in easy to access manner – as well as front-end modules – such as visualization modules, market place and portal. A more thorough overview is given in Sect. 2. In addition, A*DAX is capable to archive a large volume of data as well as to handle (and perform on-the-fly processing) on streaming data at a decent velocity.

In order to ensure interoperability between such diverse data and different analytics algorithms/processes developed by diverse sets of developers, we faced three major challenges:

1. Data encapsulation challenge: How to ensure data owners can effortlessly publish data but ensure that their data can be processed by different algorithms

2. Algorithm encapsulation challenge: How to ensure developers can easily develop (new) algorithms that can work on different pieces of data

3. User encapsulation challenge: How to ensure users can utilize (possibly third-party) algorithms on (possibly third-party) datasets without deep technical expertise
Fig. 1 depicts the three challenges in a visual format – the relationship between data, algorithm/process and user. Data can take many physical forms – some examples of data forms will be flat file (comma-separated values files), data stream or distributed data (Hadoop data). These different data formats need to be encapsulated from both the user and the analytics algorithm/process. Again, we cannot reasonably expect the third-party developers to build algorithms/processes for all types of data we have or to cover all scenarios in which his algorithm/process is invoked. Finally, users of the analytics toolkit can be human (with varying technical expertise) and/or a computerized process (RESTful container or a standalone application). This nature of the user needs to be shielded from data and the algorithm/process.

A*DAX-Data Analytics Toolkit (or A*DAX-DAT or just DAT for short) is the analytics component of A*DAX. In order to overcome the aforementioned challenges, we design DAT as three separate components: (i) DAT framework specifications, (ii) DAT libraries and (iii) development tools and plug-ins.

The first component, the DAT framework specifications, provides a set of specifications for the developers to follow while developing libraries as well as providing various tools to read data from different sources, while the second component, the DAT libraries, is a growing set of analytics libraries following the specifications. These two components ensure that the data encapsulation challenge and algorithm encapsulation challenge are well satisfied. The third component allows developers to publish standalone and client/server based applications allowing end-users to easily utilizes the DAT libraries on their own (or data accessible to them) without a steep learning curve.

In Section II, we will provide a brief overview of A*DAX and DAT, followed by Section III, where we present the design of DAT Framework specifications. In Section IV, we will discuss implementations of DAT libraries using Apriori algorithm as a working example. Then, in Section V, we will showcase DAT development plug-ins along with their example use-cases. The related works are summarized in Section VII and this paper is concluded in Section VIII.
A. The A*DAX Data Analytics Toolkit Framework

The Data Analytics Toolkit (DAT) is the Analytics Module of the A*STAR Data Analytics and Exchange Platform. In designing DAT, we aimed it to be either a standalone toolkit or a server application with Analytics As A Service (AAAS) model in mind. We also tried to make the DAT extensible with third-party libraries. Furthermore, we leveraged on the fact that certain types of analytics applications have common abstract workflow pathways.

We designed the framework as a three-component-structure. The first component, Framework Specifications defines the guideline, to which each library member in the second component, Extensible Library, follows. The third component, Component Life-cycle Framework (Eclipse plug-in) allows new algorithms to be adapted into the Extensible Libraries. In Fig. 3, we show how a new algorithm (Algorithm Z) is adapted into a new library component (Library Z) through Component Life-cycle Framework. The Framework Specification enforces Library Z can be invoked in the same way as the existing library components (Library X and Library Y) thus users have literally no learning curve.

Fig 3: Data Analytics Toolkit Framework

III. DAT FRAMEWORK SPECIFICATIONS

The DAT Framework Specifications is the component that ensures interoperability between data, algorithm/process and users. It is the heart of DAT in the sense that the Component Life-cycle Framework relies on whether each of the modules in Extensible Library follows the Framework Specifications.

We divide the framework specifications into two types: General Analytics Framework Specifications (GAFS) and Application-specific Framework Specifications (AFS). The GAFS ensures each library can access any data source, ensures each piece of data can be processed by any library and provides a uniform way to invoke any library (baseline) regardless of the environment –a standalone application, a library, or a web service. The AFS enables the developers and users utilize common application workflows (involving one or more GAFS libraries) in order to invoke any of the AFS libraries – leading to faster development and easier access to complex and advance analytics processes. An example of Application-specific Framework is Predictive Analytics Framework, in which training and prediction are common workflows. Training workflow involves two steps: training and storing model file. AFS liberates the users from tedious boiler-plate works involving in training workflow by leveraging the common workflow among similar Predictive Analytics tasks.

A. General Analytics Framework

The first part of the DAT Framework Specifications, General Analytics Framework Specifications (GAFS), ensures the following configuration. Every analytics algorithm cum process is encapsulated as a Task, which is subdivided into Batch Task and Ad Hoc Task. An example of a Batch Task is Association Rule Mining or Regression Training while that of an Ad Hoc Task is Querying an Association Rule or Regression-based Prediction. We will present the details on these Task types in Sect. 3.A.1 and Sect. 3.A.2. The input data and output result of a Task is always of type Store. There are seven pre-defined Store types as listed in Tab. 3. General Analytics Framework also defines how exactly application container invokes a Task. For instance, a RESTful web service container can invoke an Association Rule Mining task and monitor its progress (and report back to user if requested via web) until the mining process completes.

This configuration ensures several benefits. First, it provides encapsulation of data from library developer since the library developer need to support only seven (or less) pre-defined Store types regardless of physical and logical data model of the input source the user wants to use. Second, it provides encapsulation of the algorithm from data providers as they can inherit one or more types of Store to plug their data sources into A*DAX. For example, we have developed several relational database connectors in the form of Store including PostgreSQL Store and MySQL Store, which allows any algorithm to read/write data from the relational databases. Third, by implementing one of the two pre-defined Task types, it achieves user encapsulation – the library developer needs not care where and how the application container is deployed. Finally, it is possible to chain analytics tasks, i.e. using the output of an analytics task as input of another.

IV. APPLICATION-SPECIFIC ANALYTICS FRAMEWORK SPECIFICATIONS

Application-specific Analytics Framework Specifications (ASF) works on top of the General Analytics Framework Specifications. It leverages on the common application workflow in order to save both developer and users from tedious routines. Currently, we have developed Predictive Analytics Framework Specification as the single ASF. In the future, we will extend into other types of analytics functionalities and workflows.
A. Predictive Analytics Framework Specification

Predictive analytics tasks are very common analytics tasks. A common development life-cycle of a predictive analytics application is shown in Fig. 4 along with how it can be implemented using GAFS (for brevity, we omit Control, Configuration and Parameter objects). First, the developer tries to train a model using Training Data as input to the Trainer Task. The output of this process is an un-tested Model object, which is saved into a permanent storage. Then, the Model is tested with Testing Data in Verifier Task, whose output is Accuracy Results. Based on the Accuracy Results, it may be necessary to re-train the model. Using the GAFS, it involves more boiler-plate works than actual works. For instance, Verifier needs two input (Model and Testing Data) while GAFS dictates a Task can only accepts a single Store as input – requiring to have the two input Store objects nested into a container Store. Such boiler-plate works can be handled by a higher-level Framework Specifications, in this case, the Predictive Analytics Framework Specifications. Likewise, the Predictor also accepts two inputs in order to perform prediction. We provide some implementations of predictive analytics for reference as well as building block tasks (such as reading/writing from/to comma-separated value files and relational databases). Along with the Component Life-cycle Framework tools, a developer can readily build a complete predictive analytics applications in no time.

![Flowchart of Predictive Analytics Application](image)

Tab. 1 tabulates the various utilities we implemented as building blocks for both our and third-party algorithms. Most of the libraries are in-memory Store types, which is used to pass one in-memory object from one Task to the next, and data input/output utilities which convert data in various data sources to the aforementioned in-memory Stores and vice versa. The data convertors (Readers and Writers) are implemented as both Ad Hoc Task and Batch Task to allow the user to choose which method to call depending on the size of the data he is dealing with.

### Table 1. Algorithms and Utilities Implemented

<table>
<thead>
<tr>
<th>Name</th>
<th>DAT Type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSVReader</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to read a comma-separated value text file (as CSVStore) into memory</td>
</tr>
<tr>
<td>CSVWriter</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to write data in memory into a comma-separate value text file (as CSVStore)</td>
</tr>
<tr>
<td>MongoDBReader</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to read data from MongoDB</td>
</tr>
<tr>
<td>MongoDBWriter</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to write data to MongoDB</td>
</tr>
<tr>
<td>MySQLReader</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to read data from MySQL</td>
</tr>
<tr>
<td>MySQLWriter</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to write data to MySQL</td>
</tr>
<tr>
<td>PostgresReader</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to read data from PostgreSQL</td>
</tr>
<tr>
<td>PostgresWriter</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to write data to PostgreSQL</td>
</tr>
<tr>
<td>SQLServerReader</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to read data from SQLServer</td>
</tr>
<tr>
<td>SQLServerWriter</td>
<td>Ad Hoc Task/Batch</td>
<td>Used to write data to SQLServer</td>
</tr>
<tr>
<td>MemListStore</td>
<td>Store</td>
<td>Used to hold a list in memory</td>
</tr>
<tr>
<td>MemKVStore</td>
<td>Store</td>
<td>Used to hold a set of Key-value pairs in memory</td>
</tr>
<tr>
<td>MemKVListStore</td>
<td>Store</td>
<td>Used to hold a set of sorted Key-value pairs in memory</td>
</tr>
<tr>
<td>MemNDimArrayStore</td>
<td>Store</td>
<td>Used to hold a n-dimensional array in memory</td>
</tr>
<tr>
<td>CSVStore</td>
<td>Store</td>
<td>Used to hold a tabular data as Comma-separated-value file</td>
</tr>
</tbody>
</table>

V. IMPLEMENTATIONS OF DAT LIBRARIES

We implemented several example analytics (both non-predictive and predictive) algorithms as well as utilities, which can be used to build more analytics algorithms. We have implemented Apriori [3] and ECLAT [4] according to the General Analytics Framework Specifications, while Linear Regression and Back Propagation algorithms, according to the Predictive Analytics Framework Specifications. For the interest of space, we will describe only the implementation details of the Apriori algorithm in Sect. 4A.

A. Implementation of Apriori Algorithm

Apriori algorithm is a classic data mining algorithm used to find frequent item-sets. Frequent Item-set Mining problem is, given a transaction dataset, to produce a list of Item-sets containing items occurring together in transactions frequently (using a frequency threshold). It requires a single parameter, the frequency threshold (called support). Usual input of Apriori Task is a transaction table (usually a comma-separated value file), while its usual output is a list of frequent item-sets (also often a comma-separated value file).
We followed the vertical approach found in [6] to build our version of the Apriori algorithm. This requires the input of the Apriori task should be in vertical format <item_id, List(transaction_ids)>-. We develop our Apriori Task as a batch task accepting a MemKVStore as input (failing gracefully for other input types) and outputting a CSV Store containing the frequent item-sets. A usual workflow to invoke Apriori Task is a two-step workflow as depicted in Fig. 5. The first utility task, VertDB Converter, converts a transaction table (CSVStore) into a MemKVStore holding item_id to MemListStore pairs (each MemListStore hold the list of transaction_ids) so that it can be readily fed into the main Apriori algorithm. The second task, Apriori Task, is the actual Apriori processing, which outputs a CSVStore, containing the Frequent Item-sets.

In order to use Apriori Task with other input Store types, the user/developer simply needs to change the first process so that it converts his desire data-source into a MemKVStore. Conversion of output format from comma-separate value files into another format can be easily performed by fixing another conversion task as the third step.

VI. DAT TOOLS AND THEIR EXAMPLE USE-CASES

We have developed several tools for DAT development and usage including RESTful web interface for DAT libraries and Development Plug-ins, both of which encapsulate the essence of the Component Life-cycle Framework. Accompanying them we also developed some sample analytics applications which can be accessible through an iOS app.

A. DAT-REST – RESTful web interface for DAT libraries

Since it is not practical to manually maintain a RESTful web application that invokes DAT libraries, which are evolving with both first-party and third-party libraries, we need it to dynamically discover and invoke the DAT libraries without manual intervention. RESTful web interface for DAT libraries, or DAT-REST for short, is an implementation of the Component Life-cycle Framework (another implementation is the Development Plug-ins, which we will describe shortly later) in Spring Boot framework [7]. It follows the DAT Framework Specifications, i.e. it can discover DAT libraries and, assuming the role of a user, invoke them accordingly, in order to allow calling DAT libraries as an http request. Layout of the intended usage of DAT-REST is depicted in Fig. 6 below.

B. DAT Development Plug-ins (Eclipse Plug-ins)

To ease the development and development life-cycle of an analytics application project, we make an Eclipse plug-in available. It allows creation of an analytics project with the interfaces and classes of DAT Framework Specifications imported by default. It also provides widgets to create both GAFS classes (Task classes and Store classes etc.) and PFS (Trainer class, Verifier class and Predictor class etc.) along with functionalities to export into library or web applications. Fig. 9 shows the user interface to create a new analytics application project using DAT Development Plug-ins.

C. Sample Applications of DAT Libraries

We developed two working sample analytics for demonstration the power of A*DAX platform. The first sample application is retail sales analytics using Linear Regression. In this application, Linear Regression is developed using DAT Framework Specifications and exported using DAT Development Plug-in as a library, which is ultimately plugged in to DAT-REST. The application running on iOS platform accesses data (and analytics results – even ad hoc on-demand analytics results) through DAT-REST.
This configuration enables a very fast response time for otherwise lengthy (if run natively) analytics tasks as it leverages the computing power of the server. A screen from the sample application is shown in Fig. 8. The second application is analytics using Apriori algorithm.

VII. RELATED WORKS

In the context of urban analytics, large quantity and varied data are collected which requires the power of Big Data to extract the valuable insights. This will inevitably utilize complex analytics functions and primitives developed for domain-specific use-cases. Without a suitable framework to streamline the processes of Big Data analytics, the final deliverable tends to be non-reusable and non-interoperable. Apache Mahout [10] is an extensible framework for building scalable analytics algorithms. The algorithms can run on Spark 1.3+, Flink 1.0.1, and some on H2O. It involves a lot of boiler-plate codes to develop analytics algorithms and integration efforts is required with other engines (Spark and H2O) for deployment. Revolution R – Rhadoop [11] is a collection of R packages for connecting R to Hadoop and running R on Hadoop nodes. Unlike DAX, it does not streamline invocation and development of analytics functions.

VIII. CONCLUSIONS

In this paper, we presented the design and implementation of A*DAX DAT, the data analytics component of A*DAX platform. We faced data encapsulation challenge, algorithm encapsulation challenge and user encapsulation challenge in designing DAT. We overcame these challenges by designing DAT not as a static library but as a dynamic and evolving set of libraries, which adhere to DAT Framework Specifications. To ease third-party developers’ development efforts, we also implemented a RESTful interface for all DAT libraries, DAT-REST, and a development plug-in for Eclipse platform. Lastly, we presented two sample applications we developed using DAT Framework Specifications. Looking forward, we are going to extend the Framework Specifications (especially the Application-specific Framework Specifications) into new analytics territories such as Association Rule Mining and Spatial-temporal Data Analytics. We are also going to extend DAT libraries and sample applications while perfecting the DAT-REST and the development plug-in.

ACKNOWLEDGMENT

The project was supported by the A*STAR SERC Grant Project No: 1524100032.

REFERENCES

[8] Lim, P.M.C., Ng, S.-K., Ng, W.S., A*DAX: Data Analytics Platform for Smart Cities. Smart City Expo World Congress 2014