# YZStack: Provisioning Customizable Solution for Big Data

Sai Wu [‡1], Chun Chen [‡2], Gang Chen [‡3], Ke Chen [‡4], Lidan Shou [‡5], Hui Cao [#6], He Bai [*7]

[#]College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[‡]yzBigData Co., Ltd., Room 205, Tower B, Hangzhou Narada Grand Hotel, Hangzhou, China
[*]City Cloud Technology (Hangzhou) Co., Ltd.,Jiangnan Avenue, Binjiang District, Hangzhou, China

[1,2,3,4,5]{wusai, chenc, cg, chenk, should}@zju.edu.cn [6]caohui@yzbigdata.com
[7]baihe@citycloud.com.cn

## ABSTRACT

YZStack is our developing solution which implements many well-established big data techniques as selectable modules and allows users to customize their systems as a process of module selection. In particular, it includes an openstack based IaaS (Infrastructure as a Service) layer, a distributed file system based DaaS (Data as a Service) layer, a PaaS (Platform as a Service) layer equipped with parallel processing techniques and a SaaS (Software as a Service) layer with popular data analytic algorithms. Layers of YZStack are loosely connected, so that customization of one layer does not affect the other layers and their interactions. In this paper, we use a smart financial system developed for the Zhejiang Provincial Department of Finance to demonstrate how to leverage YZStack to speed up the implementation of big data system. We also introduce two popular applications of the financial system, economic prediction and detection of improper payment.

## 1. INTRODUCTION

Big data applications have drawn significant interests from both Chinese government and companies. They are keen on adopting those disruptive technologies to improve their existing services or discover the possibility of starting a completely new service. However, based on our experiences with Chinese customers, the progress of embracing the big data era is very slow due to the "3H" problem described as below:

1. *How can I build and deploy a big data system without background knowledge?* Most of our customers (e.g., government) are not IT experts. Although they know the potential benefit of big data systems, they have no idea of which systems can do what and how the system can be used to support their applications. Even deploying a Hadoop[1]+Hbase[2] environment is not a trivial task for them. What they want is something like installing a software on Windows. Users just need a few clicks and some basic customizations to make a software work.

2. *How can I migrate existing applications to the big data system?* Our customers have already maintained some IT systems to support their existing applications. Once the big data system is ready, they want to migrate those applications to the new system with few or no additional efforts, because they do not have the programmers to rewrite the applications using big data "language" (e.g., translating the codes into MapReduce [4] programs).

3. *How can I use my big data system to do the analysis job?* Our customers, such as China Southern Power Grid[3] and Zhejiang Provincial Department of Finance[4], have collected more than 20 years of data. However, they never perform any analysis for their historical data. The reason is twofold. First, they do not have proper tools to analyze such a large scale of data. Second, they do not know what can be discovered from their data and hence, cannot even start building their analytic models. A visualization tool that can be used to explore the data is currently what they want most.

Moreover, most existing products focus on a specific layer of big data ecosystem. E.g., openstack [5] and cloudstack [6] provide visualization service in the IaaS (Infrastructure as a Service) layer. Hadoop, epiC [9] and Spark [15] are popular processing frameworks in the PaaS (Platform as a Service) layer. Amazon RDS[7] is one of the popular DaaSs (Data as a Service). Linking those products together as a full-fledged big data system is a very time-consuming job, especially for customers with little big data knowledge.

To address the "3H" problem, we develop the *YZStack*, a customizable big data solution tailored for Chinese customers which covers different layers of the big data ecosystem. YZStack includes many well-established processing technologies and offers rich customization options. Users can customize YZStack by adaptively selecting building blocks for each layer. For instance, to recognize vehicle license plates from images taken by the traffic cameras, the IaaS layer establishes a cluster which consists of 100 nodes equipped with 2-core CPU, 16G memory and 256G disk. In the DaaS layer, we select the HBase as our storage system, while in the PaaS layer, we employ Hadoop as our processing engine. Finally, in the SaaS layer, we select the image-processing algorithm and data mining algorithm to process the data.

[1]http://hadoop.apache.org/
[2]http://hbase.apache.org/

[3]http://eng.csg.cn/
[4]http://www.zjczt.gov.cn/
[5]https://www.openstack.org
[6]http://cloudstack.apache.org/
[7]https://aws.amazon.com/rds/

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.
*Proceedings of the VLDB Endowment,* Vol. 7, No. 13
Copyright 2014 VLDB Endowment 2150-8097/14/08.

As shown in the example, YZStack simplifies the development of big data applications as a process of module selection and customization. However, such flexibility is not easy to achieve and it may also affect the performance. To balance the tradeoff between flexibility and performance, YZStack follows two design rules:

First, each layer defines a set of common interfaces that all its modules should implement, so YZStack can link different layers using the same codes. As each layer may include many different modules, it is challenging to design some general interfaces. Our intuition is to maintain only the fundamental interfaces. For example, in the IaaS layer, only one interface is exposed to allow users to create a cluster with virtual servers of different configurations. Configurations are passed to the interface as parameters. In the DaaS layer, a *get* interface is provided which may be explained differently by modules. The key-value store implements it as retrieving data by keys, while the relational data service uses it to return tuples satisfying query predicates.

Second, common interfaces normally provide sub-optimal performances. To further improve the performance of the system, we implement some optimization schemes as plugins in YZStack. Each optimization plugin can register for multiple modules through their *hook* interfaces to accept and process specific events. In current implementation of YZStack, we have already included some predefined optimization plugins. More plugins are being added based on users' requirement, and users can even develop their own plugins using the provided interface.

YZStack is used by our customers (e.g., CityCloud[8] and Zhejiang Provincial Department of Finance) to set up big data systems to migrate their existing applications or develop new applications. In the remaining of the paper, we will introduce the architecture of YZStack and some of its implementation details in Section 2. In Section 3, we show how we leverage YZStack to build the smart financial system for Zhejiang Provincial Department of Finance. We discuss the development of two applications on top of the smart financial system, economic prediction and detection of improper payment. Finally, we conclude the paper in Section 5.

## 2. YZSTACK: A SYSTEM OVERVIEW

YZStack is designed to simplify the development and usage of big data systems. It starts from the IaaS layer to allow users to set up a cluster from scratch to the SaaS layer, where users can develop their own applications using the provided interfaces. YZStack has been successfully used to support smart traffic, smart finance and e-card service in Hangzhou city. Figure 1 shows the architecture of YZStack which consists of four layers.

The IaaS layer is built on top of openstack with some customized implementations. We classify the resources into computation resources and storage resources which include different types of C-PUs and storage medias respectively. Combining two types of resources, we provide more than 200 instances of virtual servers. E.g., users can claim an instance with 4-core 2.6Ghz CPU, 16G memory and 256G SSD or 2-core 2.0Ghz CPU, 4G memory and 1T HDD. Computation resources are connected with the storage resources with our high speed cloud network. When creating an instance, we try to first match hardware requirement with a physical node. If the physical node can provide the necessary computation and storage resource, we directly start a virtual instance on the n-ode. Otherwise, we establish an instance with remote storage.

In the DaaS layer, on top of the DFS (Distributed File System), we provide various storage models to handle both relational data and unstructured data. The relational model can be configured as

[8]http://www.citycloud.com.cn

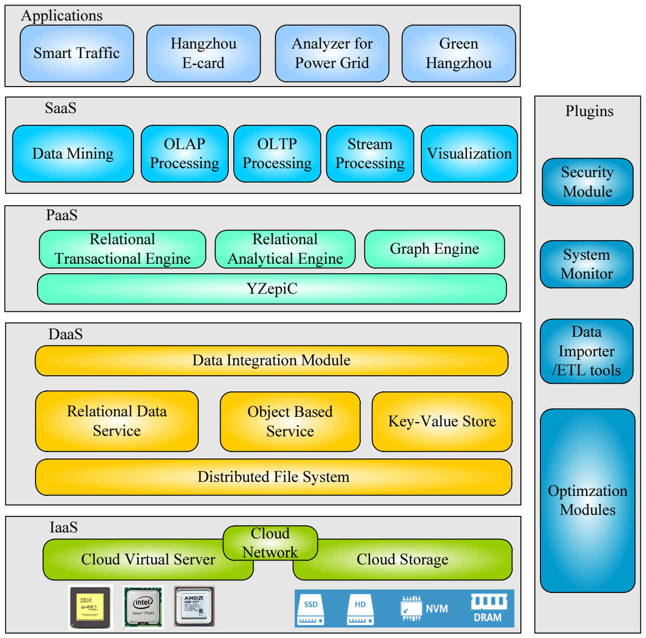

**Figure 1: Architecture of YZStack**

either row-oriented or column-oriented. A data integration module connecting multiple storage models is also included to support the processing of mixed data types.

Our PaaS layer is built on top of *YZepiC*, a parallel processing engine that implements and extends the ideas proposed in the epiC [9]. Similar as epiC, YZepiC adopts the actor model where each node in YZepiC is considered as an actor which has a job queue and can communicate with other actors via "emails". YZepiC provides a flexible programming interface in that it can simulate both MapReduce [4] and Pregel [13] model with a better performance by reusing the computation resource. Besides the basic features of epiC, YZepiC also implements some techniques which are originally proposed for optimizing the processing in the MapReduce systems [5]. To handle different data types, we develop three specific engines in YZepiC. Graph engine simulates Pregel's vertex-centric model. Relational analytic engine transforms the SQL queries into data flows between YZepiC's actors. Relational transactional engine consisting of a 2-phase locking module, a deadlock detect module and a transaction recovery module supports high-throughput distributed transactions. More details of YZepiC can be found in Section 3.2.

To simplify the development of big data applications, we implement some popular algorithms and tools as services in the SaaS layer, such as data mining algorithms, OLAP algorithms and visualization tools. Based on our customer's requirement, more algorithms are being developed and added to this layer.

Finally, YZStack creates some plugins which may interact with more than one layers. Those plugins help import the data, monitor the status of the cluster, and protect the privacy of users. Among all plugins, the optimization plugins are the most important ones which optimize the performance of the whole cluster by modifying the modules of different layers. Normally, an optimization plugin is specially designed for some module combinations. So when those modules are used, the optimization plugin is triggered.

In what follows, we introduce some implementation details in YZStack which are motivated by our customers and distinguish

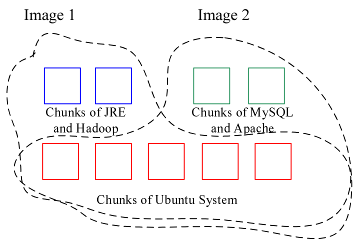

**Figure 2: Shared Chunks Between Images**

YZStack from other big data systems.

## 2.1 Adaptive Images

In the IaaS layer, when we create a virtual server for users, we actually allocate the compute resources and install a pre-generated image from the openstack. An image is a snapshot of a running system, possibly including an operating system and some popular softwares (e.g., 'Centos-with-jre" and "Ubuntu-with-mysql"). Openstack provides a tool to build customized images. To reduce the storage cost of images, we apply the chunk-based sharing approach. Figure 2 illustrates the idea, where two images share the same operating system. If we maintain the images separately, we actually replicate the data of operating system twice. So we partition the data of operating system and softwares into equal-size chunks. A unique signature is generated for each chunk. Hence, chunks with an identical signature are considered as redundant and only one copy is preserved. This strategy effectively reduces the storage overhead of images by 65%.

The image tool significantly simplifies the deployment of a cluster system. However, the problem is that we cannot pre-generate all possible images. As mentioned previously, our IaaS layer provides more than 200 types of hardware configurations. If we iterate through all possible combinations across all layers, we end up with too many images. Therefore, YZStack adaptively decides how the images are created. There are two basic image strategies.

1. The first strategy is to include as few modules as possible in the image. The image only contains the operating system and some necessary softwares. If we customize a big data system using this type of image, we need to broadcast our codes and configurations of each selected module to all virtual servers in an adhoc way. If a virtual server fails, we need to start a new one with the image and install those modules on the fly. This strategy allows us to only maintain a few images, but complicates the system setup and recovery.

2. The second strategy is to include all used modules and their configurations in one image. When a virtual server fails, we just restart a new server and install the image. No extra configuration is required. This strategy creates a static snapshot of the system. If we upgrade any module used in the image, we need to generate a new image to replace the old one.

In YZStack, we merge the two strategies to balance the update cost and installation cost. In particular, we rank modules based on their update frequency, update cost and adhoc installation cost. Modules that are frequently updated are discarded from the image, while modules incurring high installation and deployment overhead are packed into the image.

## 2.2 Optimization Plugins

The common interfaces reduce the complexity of system development, but they may also degrade the performance. In YZStack, we implement our optimization techniques as plugins which normally interact with multiple layers. Figure 3 shows how optimization plugins work with other modules. Inside each module, we have our default implementation which supports the common interface without any sophisticated technique. All modules provide a *hook* interface which can be used by the plugins to override the default processing functions with their customized ones. Once the module accepts a request defined by the common interface, instead of applying the default implementation, the module invokes the functions provided by the plugins. On the other hand, plugins can also use the cross-reference technique to access data structures of the module.

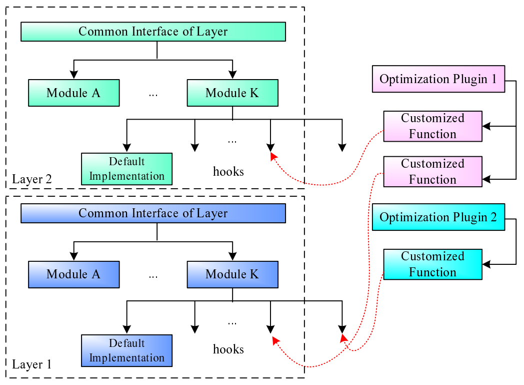

**Figure 3: Optimization Plugins**

In current implementation, we include many recently published big data techniques in our plugins which are summarized as below.

### 2.2.1 *Column-Oriented Plugin*

Column-oriented plugin is registered for the relational analytical engine. It consists of a CFile module, a compression module and a processing module. CFile is used in Llama [10] to store relational data by columns. We reuse its code to support column-oriented storage in YZStack. In the compression module, besides the popular compression algorithms such as Lzo, Gzip and Bzip2, we also implement the bitmap compression proposed in [12]. In current implementation, users should explicitly specify the compression algorithm for each column.

The processing module is applied to both the DaaS layer and PaaS layer. In the DaaS layer, it changes the default file type of the relational data service into CFile which provides basic access methods to the relational data (e.g., scan the table or given a key, return the corresponding tuple). It also provides new column-oriented access methods, such as scanning a column and retrieving the value of a specific column for a tuple.

In the PaaS layer, the processing module adds new algorithms for processing relational operators to the relational analytic engine. For example, the *Project* and *Select* operator indicate which columns should be scanned, and early materialization and late materialization are used adaptively based on the selectivity of predicate. For more details about column-oriented processing, please refer to the Llama paper [10].

### 2.2.2 *Index Plugin*

The index plugin is triggered when the relational data service is used in the DaaS layer. By default YZStack already sorts and partitions the data by primary key. Therefore, the index plugin mainly targets at the secondary index. The design of index plugin follows the same idea proposed in [11], except that we use YZepiC as our processing engine to build and use the index, instead of Hadoop. The plugin provides a GiST [8] interface. To create a new index type, the user needs to implement all methods defined in the GiST interface. Afterwards, the plugin automatically invokes those methods in building and searching the index. Currently, we have implemented B-tree, R-tree and M-tree [3] index in the plugin.

#### 2.2.3 *Query Optimization Plugin*

Our relational analytic engine works in a similar way as Hive[9] which transforms SQL queries into YZepiC jobs in an intuitive way, resulting in sub-optimal query plans. So we adopt the approach proposed in AQUA [14] by embedding an query optimizer as a plugin. The plugin first runs some YZepiC jobs to collect the statistics of data distribution. For the incoming queries, it employs a cost model to estimate the costs of different query plans. In most cases, parallel processing prefers to bushy plans rather than left-deep plans. Different from Hadoop which needs to generate multiple MapReduce jobs for a complex query, YZepiC engine is more efficient in that it only requires one job to process a query, reducing the I/O cost.

#### 2.2.4 *Iterative Job Plugin*

Many data ming algorithms, such as PageRank and LDA (Latent Dirichlet Allocation), are handled by an iterative job of YZepiC which repeatedly scans the data. To reduce the I/O costs of each iteration, we embed an iterative job plugin. The plugin defines the states of an actor in YZepiC as the data maintained or generated by the actor during its processing. It classifies the states into static states and dynamic states. Static states do not change as the computation is performed iteratively. For example, in PageRank algorithm, the graph data are static states of the actors, while the PageRank values are dynamic states. The static states are fully or partially buffered in memory (depending on the size of available memory) by actors. So in next iteration, actors do not need to recover those states. This strategy effectively improves the performance of PageRank job by almost 200%.

### 2.3 Visualization Tools

We provide a set of browser-based visualization tools which can be classified into two types: basic tools and advanced tools. Basic tools are used to produce simple charts like bar char and pie char, while advanced tools can generate social graphs, maps, and other complex diagrams. Among the advanced tools, the zoom-in visualization tool is one of the most important features of YZStack.

Our experience shows that there is normally a mismatch of anticipation between the end customers and system deliverables. Although we have provided the data mining tools, OLAP tools and other analytic tools, the customers still do not know how to leverage them to process their data. One popular question is whether we can help them use those tools to uncover the hidden value in their data. However, without specific domain knowledge, it is difficult for us to customize the analysis for their demands.

To narrow the gap, we provide a zoom-in visualization tool. The idea is to first show the big picture to users, so they can identify the interesting part and zoom in to perform a more comprehensive analysis. The zoom-in tool is linked to the background YZepiC engine and the corresponding algorithms. Each interaction between users and the tool triggers a new job processing billions of tuples.

Given a data source $\mathcal{D}$, the interface of zoom-in tool is a triple $\mathcal{T} = (\Theta, \Lambda, \Gamma)$, where $\Theta$ defines a concept set, $\Lambda$ is a set of analysis algorithms that can be applied to subsets of $\mathcal{D}$ and $\Gamma$ represents the navigation operators. $\Theta$ is organized as a tree or forest structure and the relationships between concepts are defined by $\Gamma$ as

$$\forall \gamma \in \Gamma, \forall \theta \in \Theta \Rightarrow \exists \Theta' \subseteq \Theta \wedge \Theta' = \gamma(\theta)$$

A projection operator $\odot$ is also defined for concept $\theta$ and data source $\mathcal{D}$. In particular, $\odot(\theta, \mathcal{D})$ returns a data set $S \subseteq \mathcal{D}$, and for any tuple in $S$, it satisfies the concept $\theta$.

Let $\{\theta_1, ..., \theta_k\}$ be the top concepts in $\Theta$. Initially, $\mathcal{D}$ is partitioned into subsets $\{\odot(\theta_1, \mathcal{D}), ..., \odot(\theta_k, \mathcal{D})\}$. All operators in $\Lambda$ are applied to each subset to produce the analysis results which are visualized to users. Users can select one specific concept and its subset to view the results. He can either zoom in the analysis by using the navigation operators defined in $\Gamma$. Consequently, a set of new concepts are generated and we redo the analysis for the new concepts and their subsets.

As an example, we define $\Theta$ as the time concepts, $\Lambda$ as {avg, sum, var}, and $\Gamma$ as {drill down, drill up} for the OLAP application respectively. So we can generate statistics of the database for any granularity of time (year, month or day). Another example is in the data mining context, where we can define $\Theta$ as the the wordnet's concepts, $\Lambda$ as the frequent pattern mining algorithm and $\Gamma$ as the {getparent, getchild} operations to provide more accurate pattern mining results for the documents.

The functionality of zoom-in visualization tool is still very limited. Our recent work focus on studying how the user's query can be visualized more naturally and how the results can be presented to inspire users. The tool has already drawn attention from our customers and is praised for its ability to discover the hidden patterns.

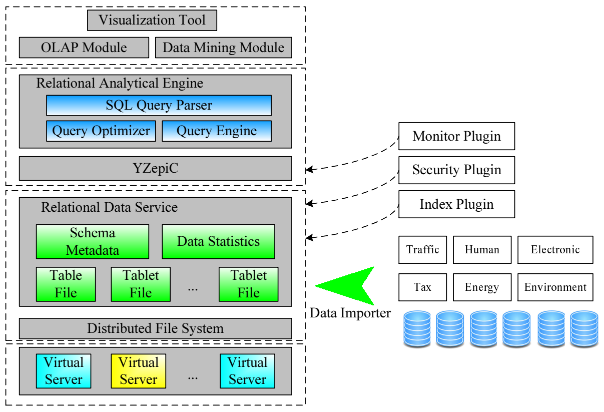

**Figure 4: Smart Financial System**

## 3. SMART FINANCIAL SYSTEM

YZStack has been commercialized as a customizable solution for big data. We collaborate with our Chinese customers to build various big data systems. One successful use case is the smart financial system, a system built for the Zhejiang Provincial Department of Finance (ZPDF) which is the financial data center of Zhejing Province that connects to many other government departments, such as the electronic department, traffic department and tax department.

Figure 4 shows how we customize YZStack to build the smart financial system. In the IaaS layer, we establish a cluster with 20 high-end virtual servers which is capable of handling current dataset (about 15TB data), and we can elastically scale up the cluster when more processing resources are required. Because the smart financial system is used as a data warehouse to analyze the

[9]https://hive.apache.org/

financial data where other departments periodically export their financial data into the system, in the DaaS and PaaS layer, we only keep the relational data service and analytical engine. We do not need to support the unstructured data and the real-time OLTP processing. In the top SaaS layer, various visualization tools are provided for users to explore the financial data. Interactions with the visualization tools are handled by the OLAP and data mining algorithms which are further translated into YZepiC jobs for processing. In this section, we discuss two applications being developed on top of the smart financial system.

## 3.1 Economic Prediction

Based all collected financial data, ZPDF wants to summarize and predict the macroeconomic trend of Zhejiang Province. Although this is a very challenging job, it has been shown previously that big data analysis can provide a very accurate economic view. The Billion Prices Project[10] at MIT successfully predicts the deflation trend during the economic crisis from 2008 to 2009 by collecting the daily prices of more than 500K products. Because ZPDF owns high-quality financial data, it believes that we can provide a more insightful view of the macroeconomics using the data. It wants us to predict the trends of economics and help identify the potential problems.

To provide an accurate prediction, we are collaborating with the researchers from college of economics, Zhejiang University. The analysis is conducted as an iterative process where our economic experts build the analytic models and we help them implement the models on the system. The analytic results are returned to the economic experts for verification who may adjust the models and repeat the above process. Currently, the analysis is conducted in three steps:

First, we use the OLAP module to provide a basic view for each registered company. A data cube is generated for data from ZPDF and Zhejiang Provincial Administration for Industry and Commerce. The data cube provides the statistics about the company's income and expenses. Users can view the results by varying the time and category dimension. Moreover, the real spending of a company is compared to its financial budget, indicating whether the company is expanding its business or shrinking it. All results in step 1 only show the financial status of an individual company which are used as the input in the following steps.

In the second step, we are using models provided by our economic experts to summarize financial reports of the companies from the same industry. There are three models being developed:

**Healthy Model** Based on the historical data, the healthy model discovers risks and predicts prospects of an industry. It clusters companies by their business scopes. As large companies are involved in a variety of business operations, they can be assigned into multiple clusters. The model runs predefined analysis for each cluster to compute the aggregate results such as total revenue, total profit and other economic indicators. The aggregate results are then fed to the model gain to generate predictions for the whole industry.

**Energy Consumption Model** We link the financial data with the electronic, water, and environment data to rank each industry based on its energy consumption per unit of output value. We also estimate its impact on the environment using the data from Zhejiang Environmental Protection Bureau.

**Economic Impact Model** By connecting the financial data to the human resource data, we study how many workers are employed for an industry and their average salary. Based on the supply chain relationship, we build a network to connect the upstream industries to the downstream ones. Using this network, we compute how a specific industry affects the others. In particular, we generate an impact index for all industries.

The last job of step two is to combine all three models to rank all industries accordingly. ZPDF requires us to show them which industry should be renovated and which one should be migrated out from Zhejiang province due to its low contribution to the economics and high energy consumption.

The last step is the most challenging task which tries to generate an economic index to predict the status of the whole Zhejiang Province using statistics generated by previous two steps. This is an ongoing work involving multiple complex economic models. We are working closely with our economic researchers to study the effects of different parameters extensively using our zoom-in visualization tool and its backend analysis modules.

## 3.2 Detection of Improper Payment

The smart financial system has collected data from different government departments. Previously, those departments never or seldom share their data with each other. So we find that there are many inconsistent data in the databases which are the major cause of improper payment. For example, in the property database, a person is classified as the low-income type and buys a house specially for low-and-medium wage earners. However, in the human database, he is employed by an IT company and has a very high pay. Another example is commonly found when companies apply for the government projects. One company may submit different registration files to different government departments (e.g., it registers as a high-tech company in the Department of Science, but as a labor-intensive one in the Department of Labor) to enjoy various allowances from the government. Preventing those improper payments can save more than one billion CNYs of government funding per year.

There are various types of improper payments. We are working with our economic researchers to develop a series of models to cover different improper payment cases. As the first step to detect the improper payment, we customize a big data verification model using the algorithms provided in the OLAP module and data mining module. Figure 5 illustrates the data flow of the model. Given two databases, we want to return all pairs of tuples that may refer to the same entity. Note that there are many sophisticated entity resolution algorithms [6][7]. However, they are too complicated to be parallelized, and for our datasets, they only result in less than 10% improvement in recall than our approach.

To detect improper payment from two databases, $\mathcal{D}_0$ and $\mathcal{D}_1$, we first generate two star-join queries, $Q_0$ and $Q_1$, which selectively merge the fact tables with the dimension tables. The trick is that the entities returned by $Q_0$ should not exist in the results of $Q_1$. E.g., $Q_0$ returns the high-income persons, while $Q_1$ returns the users who own a house specially for low-and-medium wage earners. Our economic researchers predefine such pairs of queries as rules which are periodically evaluated by our improper payment detection algorithm.

The algorithm works as follows. Let $T_0$ and $T_1$ be the results of $Q_0$ and $Q_1$ respectively. In the naive approach, for a tuple $t_i$ in $T_0$, we need to compare $t_i$ with every tuple in $T_1$ and return all possible matches. However, that is equivalent to an expensive cross product. Hence, we apply the LSH (Locality Sensitive Hashing) to generate $k$ hash values for each tuple from $T_0$ and $T_1$. So the tuples sharing the same hash value are considered as a candidate group. Given

[10]http://bpp.mit.edu/

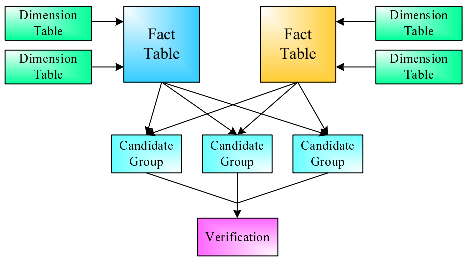

**Figure 5: Detection of Inconsistent Data**

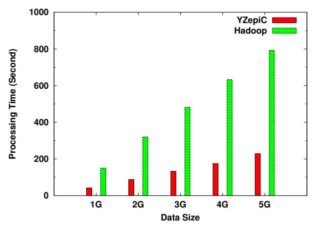

**Figure 6: YZepiC VS Hadoop**

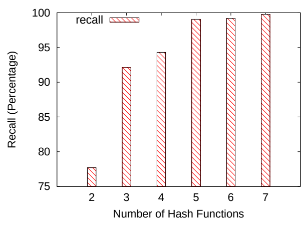

**Figure 7: Recall of Suspicious Records**

two tuples $t_i$ and $t_j$ from $T_0$ and $T_1$ respectively, $t_i$ is compared to $t_j$ only when they are from the same candidate group. We define a similarity function $sim(t_i, t_j)$ to evaluate the probability of two tuples representing the same entity. If $sim(t_i, t_j)$ is greater than a predefined threshold, it will be forwarded to the verification module where a human-aided algorithm is applied to filter out the false positives.

The selection of LSH and similarity function affects the recall and precision of the result. We consider each tuple as a small document with a set of keywords. Two sets of configurations are currently supported. Users can either use minhash [1] and Jaccard similarity or random projection hash [2] and cosine similarity. Both configurations are supported by the data mining modules and in our test, they result in a similar recall.

In fact, the model in Figure 5 is a perfect match for YZepiC. In YZepiC, we create one specific type of actors for each task. All actors are connected as a DAG (Directed Acyclic Graph) where data flow between different types of actors. Only one YZepiC job is required to process the data. On the contrary, if we use Hadoop, we need to generate five MapReduce jobs, among which four are submitted to process the joins and one is used to do the comparison. Figure 6 shows the performance comparison between YZepiC and Hadoop. Both systems run as a disk-based processing system, while our in-memory engine is being developed. Figure 7 shows the recall of our approach compared to the baseline approach (doing cross product between tables). In the test, we generate two synthetic databases using the tools provided by ZPDF, a human database and an employee database. Each database has one fact table and two dimension tables. Our system is deployed on top of a cluster with 20 nodes. Each node generates the same size of data (by default, 1GB) for the databases. We select the persons who claim to be low-income from the human database and compare them against the persons who earn more than 5000 CNY per month from the employee database. As shown in the figures, YZepiC achieves a much better performance than Hadoop on all datasets, and our LSH approach can return 99% of suspicious records by only using 5 hash functions.

## 4. CONCLUSION

In this paper, we present the design and implementations of YZStack system, our customizable solution for the big data applications. YZStack is tailored for the users who have little or no experience in deploying and maintaining the cloud system. It simplifies the development of a new big data application as the process of module selection and customization. To show the flexibility and usability of YZStack, we demonstrate how we build a smart financial system for the Zhejiang Provincial Department of Finance using YZStack. On top of the financial system, two applications, economic prediction and detection of improper payment, are developed to address real world problems.

## 5. REFERENCES

[1] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC*, pages 327–336, 1998.
[2] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.
[3] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435, 1997.
[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.
[5] D. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding*, 73(1):82–98, 1999.
[6] L. Getoor and A. Machanavajjhala. Entity resolution: Theory, practice &amp; open challenges. *PVLDB*, 5(12):2018–2019, 2012.
[7] L. Getoor and A. Machanavajjhala. Entity resolution for big data. In *KDD*, page 1527, 2013.
[8] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search trees for database systems. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *VLDB*, pages 562–573. Morgan Kaufmann, 1995.
[9] D. Jiang, G. Chen, B. C. Ooi, K.-L. Tan, and S. Wu. epic: an extensible and scalable system for processing big data. *PVLDB*, 7(7):541–552, 2014.
[10] Y. Lin, D. Agrawal, C. Chen, B. C. Ooi, and S. Wu. Llama: leveraging columnar storage for scalable join processing in the mapreduce framework. In *SIGMOD Conference*, pages 961–972, 2011.
[11] P. Lu, G. Chen, B. C. Ooi, H. T. Vo, and S. Wu. Scalagist: Scalable generalized search trees for mapreduce systems. Technique report, National University of Singapore, 2014.
[12] P. Lu, S. Wu, L. Shou, and K.-L. Tan. An efficient and compact indexing scheme for large-scale data store. In *ICDE*, pages 326–337, 2013.
[13] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, 2010.
[14] S. Wu, F. Li, S. Mehrotra, and B. C. Ooi. Query optimization for massively parallel data processing. In *SoCC*, page 12, 2011.
[15] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *HotCloud*, 2010.