International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:11, No:5, 2017

Regression Approach for Optimal Purchase of Hosts Cluster in Fixed Fund for Hadoop Big Data Platform

Haitao Yang, Jianming Lv, Fei Xu, Xintong Wang, Yilin Huang, Lanting Xia, Xuewu Zhu

***Abstract*—**Given a fixed fund, purchasing fewer hosts of higher capability or inversely more of lower capability is a must-be-made trade-off in practices for building a Hadoop big data platform. An exploratory study is presented for a Housing Big Data Platform project (HBDP), where typical big data computing is with SQL queries of aggregate, join, and space-time condition selections executed upon massive data from more than 10 million housing units. In HBDP, an empirical formula was introduced to predict the performance of host clusters potential for the intended typical big data computing, and it was shaped via a regression approach. With this empirical formula, it is easy to suggest an optimal cluster configuration. The investigation was based on a typical Hadoop computing ecosystem HDFS+Hive+Spark. A proper metric was raised to measure the performance of Hadoop clusters in HBDP, which was tested and compared with its predicted counterpart, on executing three kinds of typical SQL query tasks. Tests were conducted with respect to factors of CPU benchmark, memory size, virtual host division, and the number of element physical host in cluster. The research has been applied to practical cluster procurement for housing big data computing.

***Keywords*—**Hadoop platform planning, optimal cluster scheme at fixed-fund, performance empirical formula, typical SQL query tasks.

I. Scenario and Introduction

THIS work was done for a practical project, where a fixed fund (RMB900,000 yuan) was granted for purchasing a cluster of computers (physical hosts) to build a big data computing platform. The target was to best use the fixed fund to buy a cluster of hosts such that the cluster has a performance as high as possible. The cluster was to be used for big data queries of aggregate, join, and space-time scope selection (*space-time query*), where the cluster should provide basic effective storage and access services on large scale housing data.

In the above-mentioned scenarios, in terms of Hadoop cluster performance, the Optimal Procuring Hosts Plan (OPHP) was raised for Hadoop computing platform. It is an endless task to seek an optimal buy without significant simplification, since there are too many variables in packing different physical hosts into a cluster buy, that is, there are uncountable combinations of *physical hosts* afforded in a fixed fund, even a single vendor can offer many choices. To make things simple and easy for procurement and for maintenance afterword, we preferred a cluster consisted of identical physical hosts to that of different ones. Hereafter, we take this as granted: each physical host in a provision scheme from the same supplier is identical, i.e. they are from the same manufactory, of the same product with the same hardware configuration, and in the same price. And then, plainly the OPHP came down to be a trade-off along an axis with two opposite directions: to one side it was towards buying more hosts of lower capability in lower price, but to the other side, it was inversely towards fewer hosts of higher capability in higher price.

To study OPHP, we should firstly abstract parameters that might significantly affect the cluster performance. Obviously, these parameters should include: the number of physical hosts, and the memory size, CPU capacity, and virtual-host division, etc. of a physical host. Next, we should define a rational metric to quantitatively indicate the cluster performance. To this aim, we specified in advance in what kinds of big data computing situation should the measuring metric be applied for, as well as its empirical predicting formula be modeled and tested.

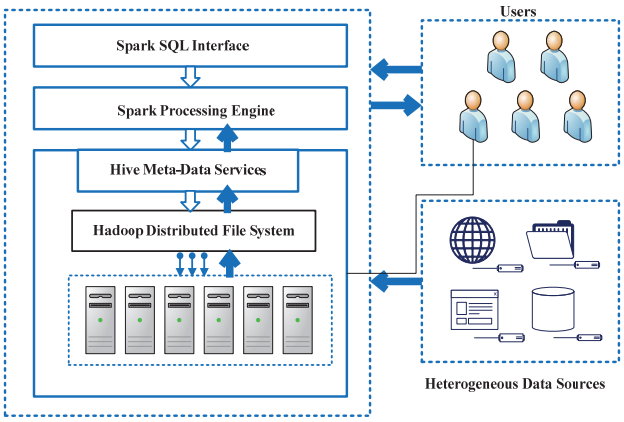

Fig. 1 HBDP's Executive Framework

H. Yang is with the Guangdong Construction Information Center, Guangzhou 510055, China (e-mail: yanght@gdcic.net).
F. Xue, Y. Huang, L. Xia, and X. Zhu are with the Guangdong Construction Information Center, Guangzhou510055, China.
J. Lv is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China (e-mail: jmlv@scut.edu.cn).
X. Wang is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China.
This research was funded by the Science and Technology Program of Guangdong Province, China, in Grant No: 2015B010131012 and No: 2016A010101012.

So far in this circumstance, the approach towards our optimal cluster purchase is still sophisticate and blurry. Hence, we had to outline a route to refine a proper right-sizing approach:

1) Construct concrete big data ecosystem –analyzing and choosing a Hadoop ecosystem (Hadoop computing context) [23] adapt to the target applications. This is with about the storage framework, data management mode, and execution context.
2) Rationally define a testable overall performance metric for a cluster of physical hosts in Hadoop computing.
3) Form an empirical predicting formula for a quantitative overall performance metric, on generally-rational principle –its coefficients can be regressed out via practical testing values, and easily used to evaluate configuration schemes of cluster hosts prior to purchase.

II. Set up Proper Big Data Computing Situation

A. Planning Ecosystem

In the background project, we were required to construct a distributed computing platform for processing massive data from housing trade and real estate register, named the Housing Big Data Platform (HBDP). In HBDP scenarios, their routine usages are mainly with storing and accessing massive relational data accumulated from historic and current housing operations, and executing typical data queries tasks mentioned in section I. Concretely, HBDP planning should be focused on the batch translating import of existent massive heterogeneous data, as well as rapid insertion and interactive queries of newly created real-time streaming data. To this aim, a Hadoop [1] computing platform was recommended.

In Hadoop computing architecture, a distributed file system HDFS is served as the bottom layer to supply basic efficient storage and storage optimization for large scale datasets. Upon HDFS, the MapReduce as a native distributed programming framework is provided as default for parallel computing.

However, the programming interface of MapReduce is rather complicated, and is hard to master for programmers unfamiliar with MapReduce programming. Hence, we need a middle layer acting just above the native MapReduce framework to provide great facility for most of data process practitioners who are familiar with SQL. According to latency requirements oriented to tasks, there are two types of this middle layer:

- That for applications which can tolerate higher latency such as batch processing of big datasets;
- That for low latency applications, e.g. interactive queries or on-line transaction processing.

As to the first type, a representative is Hive SQL computing engine [2]. This Hive SQL engine upwards provides a SQL-like program interface for programmers to code SQL style data query scripts in HQL [6], downwards offers an ETL (extract, translate, and load) data function to access large scale of datasets stored in Hadoop file system (e.g. HDFS), and is responsible for accepting HQL scripts and translating them into MapReduce jobs running on the target Hadoop cluster.

As to the latter, Apache's Spark [3] can serve well and offer a good choice. Spark's memory-based computation has a great advantage of less disk I/Os over MapReduce disk-based mode. Disk-base mode performance is poor in iteration computations, due to frequently using repetitive data and dynamically dealing intermediate data – they all require frequent external storage access I/Os, which brings with high latencies. In this aspect, Spark can improve the performance of MapReduce in iteration computing significantly. Besides, Spark might decrease the overhead of task synchronization since it uses DAG (directed acyclic graph) to optimize task scheduling.

For the sake of coding productivity and execution efficiency, we assumed Spark as HBDP interactive query framework (low latency usage). Concretely, we used Spark SQL to implement quick queries [5], [7]. The main reasons are as follows:

1) Spark SQL as an interface facility is offered by Spark for manipulating structural and semi-structural data, and its syntax is similar to that of standard SQL, which greatly simplifies higher application programming to make easy realization of interactive query analysis.
2) Spark provides high-level APIs in Java, Scala, Python and R languages, and an optimized engine that supports general execution graphs; it also supplies a rich set of higher-level tools including Spark SQL for structured data processing.
3) In HBDP, most of routine massive dataset computing is of aggregate, join, or space-time queries on relational data or structured data.

Based on the above considerations, we established a Hadoop big data computing ecosystem HDFS+Hive+Spark for HBDP.

B. Settling Executive Framework

After determining HBDP's Hadoop ecosystem, next we need to settle its executive framework for routine computing tasks. To the frame, we assumed Hadoop HDFS to store data, Hive in distributed mode to manage data, and Spark SQL to execute the so-called HBDP typical queries, i.e. quick aggregate, join, and space-time queries over massive housing data.

This settlement of HDFS+Hive+Spark ecosystem is based on their advantages and functions combination of HDFS, Hive, and Spark:

1) As well known, HDFS as a file system for elementary file storage, access and management is far from easy facilities for serving higher level data applications. Just as we need a database or data warehouse system upon fundamental file systems of operation system to make easy services for high level versatile data access we also need a similar thing upon HDFS to do so.
2) There are HBase [4] that can serve as a database system for NoSQL data access and storage, and Hive that can function as a data warehouse tool to provide interfaces of relational data modeling and processing for higher level access to data stored in HDFS. Regarding that normally routine HBDP scenarios are dealing with SQL data resources, thus we did not adopt HBase since it is not apt for relational data queries [8].
3) Spark is good at quick query computing.

In our HBDP running framework, data resources are stored in the underlying HDFS, data queries from higher applications are received, analyzed, and executed by Spark SQL on the top

layer, while Hive plays as middleware to map upper abstract data objects of Spark SQL onto lower counterparts of concrete HDFS data via meta-data interpretation, as in Fig. 1, where we just utilize Hive to manage data tables stored in external storage, rather than calling Hive's query interfaces to process data due to its high latency. Queries are executed by Spark in memories onto which involved data are loaded in the form of Hive's table.

For a practical view of HBDP application scenario, here we illustrate the most typical big data processing under the above executive framework:

At first, we shall import original massive data collected from remote heterogeneous data sources in batch into the HDFS, and build tables in Hive to store meta-data that illustrate the data to be stored in HDFS.

And then, Spark SQL component [9] will interpret SQL sentences and transfer them into RDD-based [10] query tasks when users submit aggregate, join, or space-time scope queries in SQL scripts. More concretely in the processing, HBDP reads the involved data's meta-data by Hive services, and with these meta-data it locates the corresponding target data blocks in physical storage, loads the involved data onto memory, and via *transformation* operator transfers the loaded dataset into RDD dataset in which Spark can efficiently process in parallel.

Finally, Spark will invoke corresponding *action* operator to execute query tasks in distributed mode and return computing results to users.

C. Selecting Physical Storage Mode

Regarding the above HBDP's executive mechanism as well as that HBDP's routine query tasks are carried out on memory by Spark SQL engine, we suggested that only one mandatory requirement for the procurement of the external storage devices – the volume of local hard disks of each host should exceed a desired and necessary threshold, i.e. the frequent nearby access hard disks shall have a storage volume much larger than the size of the data being involved at the calculation execution node of the Hadoop cluster. Under this condition, the configurations of host hard disk are not as sensitive and significant as the CPU benchmark, memory size, and virtual division parameters, etc. regarding the target performance of HBDP cluster.

As to concrete external storage mode scheme, there are two fundamental physical storage modes for Hadoop HDFS: local Directly Attached Storage (DAS) and Storage Area Network (SAN). We illustrated and compared them as follows.

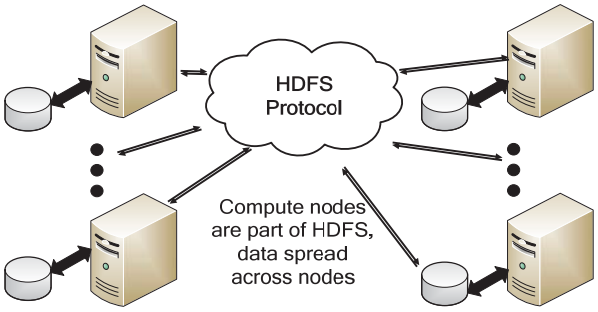

Fig. 2 DAS-based HDFS

1. Local Directly Attached Storage (DAS)

DAS is the most commonly used storage mode, as in Fig. 2. In DAS, data are spread across distributed Hadoop nodes, stored in directly-attached hard disks of hosting nodes, where each Hadoop node function as both *compute* node and *data* node, which accords with the original intention of HDFS design. In DAS, HDFS often is deployed on inexpensive common PC server hardware, which in nature bears the advantage of high lateral expansibility.

In DAS-based Hadoop clusters, fault tolerance for data storage relies on HDFS' redundant backup implementation. As soon as HDFS writes a datum in a file block it will also write the same datum in several redundant file blocks distributed in different Hadoop nodes; when HDFS reads a data block, it can access the same data block in multiple nodes in parallel at the same time, which makes broader I/O bandwidth [22].

2. Storage Area Network (SAN)

SAN is also a main storage mode for Hadoop. In SAN, data are stored concentrative in disk-array devices, as in Fig. 3. RAID implementation of storage arrays provides fundamental fault tolerance of data storage. A virtual private SAN storage space is configured to each Hadoop node attached to the SAN.

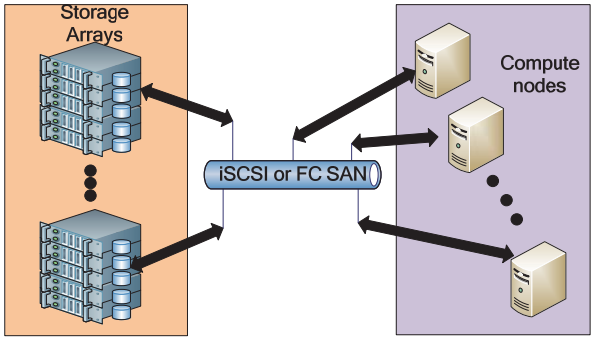

Fig. 3 SAN-based Hadoop cluster

In SAN-based Hadoop clusters, writing data in HDFS does not need to write their redundant data block backups elsewhere since data fault-tolerance is guaranteed by RAID mechanism of storage array; reading data from HDFS will not access multiple nodes, instead it will read directly from the disk-array where the data was placed [22].

The advantage of SAN comes from: RAID, cache, thin provisioning, and non-redundant writing with HDFS. SAN, however, has a relatively poor I/O paralleling capability and lateral expansibility.

3. Selection of Data Storage Mode

For selection of data storage mode, we considered the factors such as I/O bandwidth, fault tolerance, expansibility, and prices comprehensively. To sum up, we list the concerned aspects of DAS and SAN in Table I for comparison.

For greater I/O paralleling, higher lateral expansibility, and overall performance/price ratio in HBDP project, we decided to deploy HDFS in DAS mode. In fact, DAS was more in line with the original intention: "Hadoop was designed to move compute

closer to data and to make use of massive scale-out capabilities," as stated in [17].

TABLE I
ASPECT FOR COMPARISONS ON DAS AND SAN

Mode	Cost	I/O bandwidth	Fault Tolerance
DAS	inexpensive	High	By HDFS
SAN	expensive	Lower	By RAID

Although the redundant storage policy of DAS could bring about decrease in disk utility, however, this would weigh less and less as hard disk devices become increasingly cheaper. In practice, often a low threshold of hard disk volume is set for each host to guarantee their individual performance as well as decrease network data access frequency. In HBDP project, we rationally required that local hard disks of each physical host well accommodate a copy of maximum local computing data, i.e. greater than 5TB — this is common to most of today PC server. Thus, hereafter local external storage is not of the focus simply due to a premise: 1) local storage is often sufficient for each *compute* node with respect to its assigned computation; 2) each time executing computation, Spark SQL should have its object dataset loaded onto memory first, that is, the involved external hard disk access, either local or networked, has be executed once in advance.

III. MODELING AND RELATED JUSTIFICATION

After making choice of the big data computing situation, we should outline HBDP's basic model of query computation, as well as procurement regulations for physical hosts of a Hadoop cluster. And then, we conducted query tests with Spark SQL, Hive, and Oracle to validate that Spark SQL has performance advantage over the other two. Here, Oracle is a representative of classical computing engine for massive data query, while Hive is a kind of Hadoop computing engine outside our choice.

A. Tasks of Typical Queries and Its Test Data Sets

Based on the routine computing tasks of HBDP applications, to develop a predicting formula for cluster performance metric, we shall focus on HBDP's typical query tasks, i.e. join query task *T_J*, aggregate query task *T_A*, and space-time scope query task *T_S*. To this end, we construct two schemes of housing datasets in two typical scales, as in Table II.

TABLE II
HOUSING DATA INSTANCES FOR PERFORMANCE EXPERIMENTS

Instance	Scheme	Scale	Volume	Contents
Dataset_1	Housebase	Ordinary	493.18MB	$1{\cdot}10^7$ Units of house
Dataset_2	Housebase	Large	14.53GB	$5{\cdot}10^8$ Units of house
Dataset_3	Housetrans	Ordinary	583.89MB	$3{\cdot}10^7$ Records of trade
Dataset_4	Housetrans	Large	24.65GB	$5{\cdot}10^8$ Records of trade

The relational schemes in Table II are defined as follows:

1) *Housebase* (*house_id, pur_date, longitude, latitude, house_type, house_size*) scheme for house attribute data; *dataset_1* and *dataset_2* are its instances.
2) *Housetrans* (*trans_date, house_id, price, trans_ident*) scheme for house trade records; *dataset_3* and *dataset_4* are its instances.

Next, we illustrated task *T_J*, *T_A*, and *T_S* in the following examples with dataset instances from Table II:

1) Task *T_J*: joining data tables *housebase* and *housetrans* via their common field *house_id*; a sample of SQL code is as following:

```
select distinct trans_date, trans_price, house_type
from housebase left outer join housetrans
on (housebase.house_id = housetrans.house_id)
where housebase.house_id = '01000000'
```

2) Task *T_A*: calculating the total area of houses with respect to various different housing types on data table *housebase*; a sample of SQL code is as following:

```
select house_type, sum (house_size)
from housebase
group by house_type order by house_type asc
```

3) Task *T_S*: from data table *housebase* searching all house identities that satisfy a space-time condition; a sample of SQL code is as following,

```
select house_id from housebase
where longitude between 23.03 and 23.084054
and latitude between 113.18 and 113.234054
and pur_date between '20000101' and '20000601'
```

B. Shaping Cluster and its Performance Metric

Under DAS *storage* mode and Spark *compute* mode, in the sense of high probability all computation-needed datasets were once a time drawn within nearby storage (in local hard disks or even memories), regarding that all nodes in HBDP were located in the same high-speed LAN. Thus, we shaped HBDP cluster configuration in simple:

1) Each member physical host is of the same configuration in the same type, and has the same virtual hosts division.
2) The volume of each node's local hard disks, e.g. 2TB is well beyond its highest possible data size e.g. 50GB for a single query computation.

TABLE III
PARAMETERS USED IN MODELING HBDP CLUSTER

Symbol	Meaning	Metric Units
M	Physical host's memory	GB (Giga Byte)
C	Physical host's CPU benchmark	PassMarkt result [11]
N	Number of physical hosts	Number
V	Number of virtual hosts within a single physical host	Number
P	Overall performance metric of HDBP cluster	Numeral value
t_a	Elapsed time of executing task *T_A*	Second
t_j	Elapsed time of executing task *T_J*	Second
t_s	Elapsed time of executing task *T_S*	Second

Under these simplifications, hard disks' parameters of each physical host are not test factors for the cluster performance. Hence, it is applicable that the metric for cluster performance is mainly up to the memory size, CPU capability, virtual-hosts division of individual physical host, and the number of physical hosts in cluster. We used symbols of parameter as in Table III to model HBDP cluster performance.

In OPHP research, we adopted CPU benchmarks published by PassMark [11], or those measured with PassMark open tool —PassMark has published benchmark test results for more than 1200 CPU models, and provided an open measuring tool for users' self-test.

Based on the above three routine tasks of HBDP applications, we defined the performance metric for a HBDP cluster as:

$$P = 1/(t_a + t_j + t_s) \quad (1)$$

The HBDP cluster performance P is counted inversely on the elapsed time sum of executing three types of routine tasks.

Next, we had the following modeling principles for HBDP cluster performance prediction:

1) Intuitively construct an empirical formula of predicting P on variables M, C, V, and N with pending coefficients.
2) Measure the value of P defined in (1) for clusters samples of different configurations bought in the fixed fund.
3) Match the measured results of the previous step to regress all pending coefficients in the empirical formula of P.

C. Validation Tests with Spark SQL, Hive, and Oracle

To justify that the Spark SQL is the right choice of big data platform for HBDP, we conducted experiments of executing typical query tasks respectively with Spark SQL, Hive, and Oracle for performance comparison. To be fair, in the tests, we configured Spark SQL cluster, Hive cluster, and Oracle server with the same hardware resource, as in Table IV. Two large scale datasets as stated in Table III were imported for executing query tasks of *T_J*, *T_A*, and *T_S* types. For each query type, a data import job and 50 query jobs were concurrently submitted, and the time spent on data import and the execution time of all 50 concurrent jobs were recorded for comparison.

TABLE IV
HARDWARE FOR SPARK SQL, HIVE, AND ORACLE COMPUTING

	Spark SQL cluster	Hive cluster	Oracle server
Memory	3×(8 GB)	3×(8 GB)	24 GB
CPU cores	3×(1 core)	3×(1 core)	3 cores

TABLE V
TIME USED BY SPARK SQL, HIVE, AND ORACLE COMPUTING

Time used↘	Spark SQL cluster	Hive cluster	Oracle server
Import job	7min 25s	9min 53s	1h 28min 26s
T_J jobs	1min34s	5min14s	9min28s
T_A jobs	53s	1min17s	6min31s
T_S jobs	25s	1min 52s	5min1s

The results of experiments were listed as in Table V. It showed that the Spark cluster had rather greater performance advantage over the Hive cluster and the Oracle server on high concurrency jobs of large scale data disposal, and the Spark cluster and the Hive cluster both had much greater performance advantage over the Oracle server (Oracle 11g on 64bits Linux) on large scale data import. Related average usages of main computing resources in executing the above computing jobs were recorded in Table VI; it told that the Oracle server almost reached its CPU processing ceiling during the tests, while the other two platforms were still halfway to their upmost.

TABLE VI
RESOURCE USAGE COMPARISON

Resource Usage↘	Spark SQL cluster	Hive cluster	Oracle server
Memory	12.1 GB	11.6 GB	2.2 GB
CPU	51%	52%	97%
Network Bandwidth	Out: 7.4Mb/s, In:12.1Mb/s	Out: 16.9Mb/s, In:17.4Mb/s	60.5Mb/s, In:14.4Mb/s

IV. EMPIRICAL FORMULA FOR CLUSTER PERFORMANCE

The previous sections justified that our HDFS+Hive+Spark big data computing situation is appropriate for HBDP. The next steps towards an optimal purchase of hosts cluster is to shape an empirical formula to predict the value of metric *P* for cluster performance, and on regression of the pending parameters of the empirical formula by experiments with sample clusters.

A. Empirical Formula for Cluster Performance

It is time to give an empirical formula in parameters *M*, *C*, *N*, and V which are as defined in subsection III.B for predicting HBDP cluster performance benchmark. The empirical formula is established in the following steps.

1) As to a primitive physical host, we intuitively used a linear function of *M* and *C* for its performance formula:

$$P(1) = b_1 M + b_2 C \quad (2)$$

2) When this physical host is divided equally into V virtual hosts, the physical host can be regarded equivalent to a cluster of V virtual hosts, and its performance metric could be modeled as

$$P(V) = \frac{b_1 M + b_2 C}{V} f(V) \quad (3)$$

where $\frac{b_1 M + b_2 C}{V}$ stands for the performance metric value of a single virtual host, equaling to 1/*V* of the physical host's; $f(V)$ is called a *speedup* function [21], it reflects the performance amplification of a Spark cluster scaled from 1 virtual host to *V* virtual hosts. Due to sophisticated overlapping of costs from resource contentions and distributed incoherency of *V* virtual hosts, $f(V)$ is normally not linear.

3) As for a cluster of N physical hosts (each is divided into V virtual hosts), its performance benchmark should be

$$P(V, N) = \frac{b_1 M + b_2 C}{V} f(V) g(N) \quad (4)$$

where $g(N)$ is also a cluster performance *speed-up* function similar to $f(V)$ in formula (3) [21].

According to [21], the *speedup* function of the performance amplification of a cluster scaled from 1 host to x identical hosts can be expressed as

$$Sp(x) = \frac{x}{1 + \delta(x-1) + \kappa(x-1)x} \quad (5)$$

where δ reflects a degree of contention for shared resources and κ stands for incoherency of distributed data among x hosts or nodes of the cluster. Obviously, $Sp(x)$ is not linear with x, the

hosts or nodes number. Hence, the above $f(V)$ and $g(N)$ as a *speedup* function of nodes *V* or *N* respectively shall be as

$$f(V) = \frac{V}{1+b_3(V-1)+b_4(V-1)V} \quad (6)$$

$$g(N) = \frac{N}{1+b_5(N-1)+b_6(N-1)N} \quad (7)$$

and then, we have:

$$P(V,N) = \frac{(b_1M+b_2C)N}{(1+b_3(V-1)+b_4(V-1)V)(1+b_5(N-1)+b_6(N-1)N)} \quad (8)$$

B. Approaching Empirical Formula Regression

$P(V,N)$ from the predicting formula (8) should be fitted with P from the measured formula (1) for a given cluster. The target of fitting $P(V,N)$ is to adjust coefficients $\{b_1, b_2, b_3, b_4, b_5, b_6\}$ to minimize the average squares of deviations between the predicted value from formula (8) and the measured value from formula (1) for the performance metric of all clusters sampled:

$$\min \frac{1}{n}\sum_{i=1}^{i=n}\left(P_i - \hat{P}_i\right)^2 \quad (9)$$

where n is the number of clusters sampled, P_i and $\hat{P}_i$ are the measured value and the predicted respectively for *i-th* sampled cluster's performance.

Regarding that these six coefficients in the empirical formula are so broad and loose to be regressed over limited experiments, we had better refer to results from other researchers. Since the Spark-feathered HBDP running contexts could be categorized into the big-memory type in [21], we adopted corresponding two coefficients from [21] for the speedup function of *N* nodes' Hadoop cluster. That is, we took:

$$b_5 = -0.0288, b_6 = 0.000447$$

The above values of coefficients $\{b_5, b_6\}$ are regressed from abroad scope of clusters scaled up to over 48 nodes, which well covers the potential cluster scale of HBDP, and particularly, they are concerning with nodes of physical hosts, thus they are fit for adoption in formula (7) and hence (8). Together with this reference to peer research [21], we can focus on regressing coefficients $\{b_1, b_2, b_3, b_4\}$.

C. Samples of Test for Regressions Analysis

For regressing coefficients $\{b_1, b_2, b_3, b_4\}$, performance tests of various HBDP clusters are required to get samples of typical tasks' elapsed times $\{t_a, t_j, t_s\}$ for formula (1). Since computer resources available were limited - 3 PC servers with CPUs of dual Intel Xeon ™ E5-2690 (10 cores, 3.00 GHz), 200 GB RAM memory, Hard Disks of 500 GB EMC ™ VNX5400, we had to resorted to virtualization technique to simulate various samples of multiple physical and virtual hosts as test scenarios on these 3 PC servers. At first, in each PC server, we created a virtual machine as a *simulated physical host* such that its CPU cores and memory assigned can be equally divided further into smaller granularity to create virtual machines of same build as *simulated virtual hosts* for HBDP clusters. Each configuration of such simulated physical hosts and their virtual hosts were taken as a regression sample. All virtual hosts were created with VMware vSphere [12], [16], and each virtual host was mounted with an Ubuntu 14.04 LTS operation system [13]. Hadoop clusters were established by Ambari [14], and were deployed with HDP 2.4, the latest stable Hadoop ecosystem [15].

12 samples of HBDP cluster test were included in Table VII where column labels ***S***(*i*), M(G), Core (Benchmark), *N*, and *V* stand respectively for sample(*i*), the size (in Giga-bytes) of memory of each *simulated physical host*, the number of CPU cores (Benchmark on-site) of each *simulated physical host*, the number of *simulated physical hosts*, and the number of *simulated virtual hosts* in single *simulated physical host*. The elapsed times $\{t_a, t_j, t_s\}$ were recorded in seconds, which were actually the average values of 50 rounds of task executions.

TABLE VII
SAMPLES OF TEST

S(*i*)	***M***	**Core (Benchmark)**	***N***	***V***	t_a	t_j	t_s
i=1	8	1 (1527)	3	1	1.54	38.99	2.28
i =2	24	16 (17349)	2	2	0.66	2.96	1.09
i =3	24	12 (14960)	2	2	0.73	2.95	1.24
i =4	24	8 (11400)	2	2	1.12	3.54	1.4
i =5	24	4 (7226)	2	2	1.33	4.45	1.63
i =6	32	8 (11400)	2	2	0.86	3.4	1.18
i =7	36	18 (18529)	1	3	0.78	9.73	0.89
i =8	36	12 (14960)	1	3	0.96	11.24	1.02
i =9	36	12 (14960)	2	3	0.84	11.94	1.78
i =10	40	8 (11400)	2	2	1.01	3.35	1.04
i =11	48	12 (14960)	1	3	0.93	11.1	0.81
i =12	60	12 (14960)	1	3	1.08	10.5	1.22

TABLE VIII
COMPUTING ON SAMPLES

S(*i*)	$\hat{P}_i$	P_i	$\left(P_i - \hat{P}_i\right)^2$
i=1	0.0928241	0.023359	0.0048254
i =2	0.1960494	0.2123142	0.0002645
i =3	0.1789188	0.203252	0.0005921
i =4	0.1533915	0.1650165	0.0001351
i =5	0.1234613	0.1349528	0.0001321
i =6	0.1772736	0.1838235	4.29E-05
i =7	0.0634396	0.0877193	0.0005895
i =8	0.0566842	0.075643	0.0003594
i =9	0.1166229	0.0686813	0.0022984
i =10	0.2011558	0.1851852	0.0002551
i =11	0.0661403	0.0778816	0.0001379
i =12	0.0755963	0.078125	6.394E-06

D. Solving Pending Coefficients with MS Excel

Under the test scenarios sampled in the previous subsection, we used the *Solver Add-in* provided by Microsoft Excel 2016 to approach the optimal target defined in formula (9) via the GRG algorithm of nonlinear programming solving [18]-[20]. For pending $\{b_1, b_2, b_3, b_4\}$, starting with initial values {0, 0, 0, 0} we got their converged values {0.002506215, 6.01992E-06, 0.001306984, 0.36297383} that made formula (9) arrived at a least value 0.0008032 for cluster samples {cluster (*i*) | *i*=1, 12}. The computing result was listed in Table VIII where *S*(*i*) stands

for sample(i) as in Table VII, and P_i and $\hat{P}_i$ are identical as in formula (9). Hence, we determined all coefficients in formula (8):

$$b_1 = 0.002506215, b_2 = 6.01992\text{E}-06,\ b_3 = 0.001306984,\\ b_4 = 0.36297383,\ b_5 = -0.0288,\ b_6 = 0.000447.$$

E. Empirical Formula's Rationality and Simplicity

The formula (8) accords with the following expectations:

1) The larger a single physical host's memory M, the higher the $P(V,N)$, and the better the cluster performance, in the case of other parameters unchanged.
2) The larger CPU benchmark C for single physical host, the higher the $P(V,N)$, and the better the cluster performance, in the case of other parameters unchanged.
3) $P(V,N)$ decreases as V increases ($b_3, b_4 > 0$), i.e., the more the virtual hosts a single physical host is divided into, the worse the cluster performance due to the increasing overhead from resource division, when other parameters remain unchanged.
4) When V=1, N=1, formula (8) is reduced to formula (2).

According to (3), we needn't take into account any virtual host division in procuring physical hosts, i.e., for such a usage formula (8) can be simplified by taking V=1 as:

$$P = \frac{(0.00251M+0.00000602C)N}{(1-0.0288(N-1)+0.000447(N-1)N)} \quad (10)$$

TABLE IX
CLUSTER SCHEMES SURVEYED ON MARKET

Host model(Memory)	CPU model	CPU benchmark	Hard Disk	Host Amount	*P*
Lenovo-X3650M5(192G)	2×XeonE5-2650V4	22,619	2.7T	18	1579
Lenovo-X3650M5(256G)	2×XeonE5-2650V4	22,619	2.7T	16	1345
DELL-R730(128G)	2×XeonE5-2630V4	18,801	3.6T	20	1516
DELL-R730(256G)	2×XeonE5-2660V4	22,164	3.6T	16	1318
HUAWEI-RH2288H V3(256G)	2×XeonE5-2660V4	22,164	4.5T	14	1102
HUAWEI-RH2288H V3(512G)	2×XeonE5-2660V4	22,164	4.5T	10	712
HUAWEI-RH5885 V3(256G)	4×XeonE7-4830v3	*30,084	5.4T	8	734
HUAWEI-RH5885 V3(512G)	4×XeonE7-4830v3	*30,084	7.2T	6	521
Inspur-NF5280M4(256G)	2×XeonE5-2660V4	22,164	5.4T	11	804
Inspur-NF5280M4(512G)	2×XeonE5-2660V4	22,164	5.4T	8	541
Inspur-NF8460M4(128G)	4×XeonE7-4830v3	*30,084	5.4T	7	625
Inspur-NF8460M4(512G)	4×XeonE7-4830v3	*30,084	5.4T	6	521

* The benchmark of 4×XeonE7-4830v3 is a fitted value since its direct value is not available

V. EXAMPLE OF EMPIRICAL FORMULA APPLICATION

In the practice of HBDP cluster purchase, we used formula (10) to predict the cluster performance of various schemes (well fitted to the budget) available on the market of name-brand PC servers. Before calling for bidding, we got to obtain a concrete specification for the cluster hardware. Through a market survey, we collected a list of PC server provision scheme, and then work out their predicted performance values in formula (10). The evaluation result was recorded in Table IX, where we had two top cluster schemes: 18 sets of Lenovo-X3650M5 (192G), and 20 sets of DELL-R730 (128G), prominent on performance metric: scored 1579 and 1516 respectively. Regarding that the predicted performance of these two candidates were very close, we needed to make a further comparison: the Lenovo one has a clear CPU advantage (XeonE5-2650V4 over XeonE5-2630V4), and its memory is 1.5 times of the Dell's although the latter provides more physical hosts but which only had a narrow lead in host amount in a proportion of (20-18)/18=1/9. Thus, 18 sets of Lenovo-X3650M5 (192G) was recommended.

VI. CONCLUSION

Optimizing procurement of device cluster at a fixed budget is a broadly met issue, since in general device requirements are not specific enough to determine each device's configuration in details. Similar processes as the optimal purchase approach presented above might or should be done in this or other ways over and over again, since it is often a field specific tangle over that whether fewer devices of higher capability or more devices but of lower capability. We hope that this regression approach could be heuristic for similar tasks in other fields.

REFERENCES

[1] http://hadoop.apache.org/(Accessed on 07/03/2017).
[2] https://hive.apache.org/(Accessed on 07/03/2017).
[3] http://spark.apache.org/(Accessed on 07/03/2017).
[4] https://hbase.apache.org/(Accessed on 07/03/2017).
[5] http://spark.apache.org/sql/(Accessed on 07/03/2017).
[6] Capriolo E, Wampler D, and Rutherglen J, *Programming hive*. O'Reilly Media, Inc., 2012.
[7] Karau H, Konwinski A, Wendell P, et al, *Learning spark: lightning-fast big data analysis*. O'Reilly Media, Inc., 2015.
[8] http://hbase.apache.org/book.html#arch.overview(Accessed on 07/03/2017).
[9] M. Armbrust, R. S. Xin, C. Lian, et al, "Spark SQL: Relational data processing in spark," in *Proc. of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM, 2015, pp.1383–1394.
[10] M.Zaharia, M. Chowdhury, M. J. Franklin, et al, "Spark: cluster computing with working sets," in *Usenix Conference on Hot Topics in Cloud Computing*, USENIX Association, 2010, pp.1765–1773.
[11] http://www.CPUbenchmark.net/(Accessed on 07/03/2017).
[12] http://www.vmware.com/(Accessed on 07/03/2017).
[13] http://www.ubuntu.org/(Accessed on 07/03/2017).
[14] https://ambari.apache.org/(Accessed on 07/03/2017).
[15] http://hortonworks.com/products/data-center/hdp/(Accessed on 07/03/2017).

[16] B. Sotomayor, R. S. Montero, I. M. Llorente, et al, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet computing*, 2009, vol. 13, no. 5, pp. 14–22.
[17] http://www.computerweekly.com/feature/Big-data-storage-Hadoop-storage-basics(Accessed on 07/03/2017).
[18] A. M. Brown, "A step-by-step guide to non-linear regression analysis of experimental data using a Microsoft Excel spreadsheet," *Computer Methods and Programs in Biomedicine*, 2001, vol. 65, no. 3, pp. 191–200.
[19] C. L. Lawson, R. J. Hanson, *Solving least squares problems*. Society for Industrial and Applied Mathematics, 1995.
[20] M.J. Box, D. Davies, and W.H. Swann, *Non-Linear optimization Techniques*. Oliver & Boyd, 1969.
[21] N.J.Gunther, P. Puglia, K. Tomasette, "Hadoop Superlinear Scalability," *Communications of the ACM*, 2009, vol. 58, no. 4, pp. 46–55.
[22] A. Mukherjee, J. Datta, R. Jorapur, et al, "Shared disk big data analytics with apache Hadoop," *19th international conference on High Performance computing (HiPC2012)*, IEEE, 2012, pp. 1–6.
[23] T. White, *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

Haitao Yang received his Bachelor of Engineering degree in mechanics in 1983, Master of Science degree in computational mathematics in 1989, PhD degree in computer software and theory in 2008, from the National University of Defense Technology, Sun Yat-sen University, and Computing technology institute of Chinese Academy of Sciences, China, respectively.

He has been working in computer-related fields over three decades. His professional career is primarily as designer and analyst of information system and computing platform. In the recent years, his major interest focused on big data computing and Internet distributed system, and their applications in urban and rural building domain.