Computer Science and Information Systems 18(1):1–22 https://doi.org/10.2298/CSIS200131031C

Throughput Prediction based on ExtraTree for Stream Processing Tasks

Zheng Chu[1], Jiong Yu[1], and Askar Hamdulla[1]

School of Information Science and Engineering, Xinjiang University, Urumqi 830046, PR China
chuzheng@stu.xju.edu.cn,{yujiong,askar}@xju.edu.cn

Abstract. In the era of big data, as the amount of streaming data continues to increase, stream processing tasks (SPTs) face serious challenges in real-time processing scenarios with low latency and high throughput. However, much of the current literature on the performance of SPTs pays attention to the reactive approach, which cannot well avoid the problem of system crashes due to the inherent performance volatility. In this paper, a novel throughput prediction method based on ExtraTree for SPTs is presented to address these challenges. A volatility detection algorithm was proposed to obtain the reasonable metric values after the performance volatility of SPTs was studied. Moreover, a selection algorithm of regression function was proposed to output the performance values of SPTs under a relative stead state. Furthermore, a ExtraTree-based algorithm was proposed to predict the throughput of SPTs. The experimental results from two open-source benchmarks running on Apache Flink, a popular stream processing system (SPS), indicated that the average of the accuracy and efficiency of the proposed method could achieve 90.535% and 0.835 s/10,000 samples, which proved the effectiveness of the proposed method on the task of predicting the throughput of SPTs.

Keywords: streaming data, stream processing tasks, performance prediction, ensemble learning, ExtraTree.

1. Introduction

The emergence of big data processing systems enables organizations to store and process high-dimension, diverse, and high-speed data [17]. Data processing approaches are usually divided into batch processing and stream processing. The former is generally used for static data, and the latter is used for streaming data. For dynamically changing data, most of the systems based on the Map-reduce [6] computing algorithm use the batch processing approach to process and analyze the data. Products in the ecosystem include HDFS [2], HBase [25], and Hive [27]. Accordingly, popular stream processing systems (SPSs) include Apache Flink [4], Twitter Heron [16], and Apache Storm [26], etc. These systems mainly use stream processing approach to process and analyze data.

With the rapid development of social media [8], news sources, and the Internet of Things (IoT) [14], large-scale streaming data from various sensor devices [20], mobile devices [15], and smart devices [13] generated and streamlined by the SPS in real time. Due to the streaming data with the characteristics of large scale, rapid change, and continuous generation, SPSs and SPTs must ensure low latency and high throughput as much as possible. To achieve the dual goals of low latency and high throughput, current research efforts are focusing on task scheduling [29], load balancing [18], elastic computing [11], etc. In these studies, all strategies are triggered after streaming data occurs a burst and the performance of SPT cannot meet the requirements of users, i.e., reactive approach. This approach may render the system unavailable for a certain period. If the streaming data continues to fluctuate or oscillate, the above-mentioned reactive approach will cause the system to enter a continuous adjustment process, which will cause the continuous adjustment time exceed the available time, and even cause the system to crash.

A reasonable approach is to predict the throughput of SPTs under different conditions, i.e., different streaming data rates. If the current throughput can be predicted in advance, system crashes can be better prevented. To avoid this situation that SPTs cannot cope with the rapid increase in the amount of streaming data due to its limited processing capacity, the study of this paper aims to predict the maximum throughput of SPTs with latency guarantees. This issue is also an important research in load management, query scheduling, permission control, schedule monitoring, system scale customization, etc., and these studies will not be described here.

To predict the maximum throughput of SPTs with latency guarantees, we analyzed the performance volatility of SPTs and proposed a detection algorithm for performance volatility. Moreover, a polynomial regression algorithm was applied to the performance volatility analysis to accurately estimate the throughput at a specific data rate. Furthermore, the throughput of SPTs in a relatively stable state was output using a volatility regression algorithm. Finally, an ExtraTree-based algorithm was used for predicting the throughput of SPTs.

The main contributions of this paper are as follows:

(1) A volatility algorithm was proposed to detection the performance volatility of SPTs after the volatility was studied.

(2) A polynomial regression algorithm was proposed to apply to the performance volatility to evaluate the performance of SPTs in a relatively stable state by configuring different regression items and selecting the appropriate regression function automatically.

(3) An ExtraTree-based algorithm was proposed to predict the throughput of SPTs. In particular, we first predict the maximum throughput of SPTs in this paper;

(4) The experimental results from two open-source benchmarks running on Apache Flink, a popular SPS, indicated that the average of the accuracy and efficiency of the proposed method could reach 90.535% and 0.835 s/10,000 samples, which proved the effectiveness of the proposed method. More importantly, the proposed method outperforms other ensemble learning algorithms in term of accuracy and efficiency.

The structure of this paper is organized as follows: Section 2 gives an overview of the related work. Section 3 describes the performance volatility phenomenon and volatility detection algorithm in detail. The performance evaluation method for SPTs are described in Section 4. Section 5 elaborates on the performance prediction algorithm. In Section 6, we evaluate the effect of the ExtraTree-based throughput prediction algorithm on SPTs through experiments and analysis. Section 7 concludes the paper with a summary and suggestions for future works.

2. Related Work

At present, related works have achieved good results in many fields, e.g., natural language processing [30], speech recognition [7], image processing [21], autopilot [1], etc., using machine learning algorithms, while there are relatively few works on traditional computer systems, especially SPSs. In this section, we will describe related works, mainly involving SPSs, ensemble learning, and performance prediction for SPTs.

Stream processing systems: A stream processing system is a kind of system that continuously processes, aggregates, and analyzes streaming data. Unlike Hive and HBase, it is a software system based on a stream computing framework. The processing latency of a SPS is measured in seconds or even milliseconds level. Such a system typically uses a Directed Acyclic Graph (DAG) computation algorithm to process streaming data on the nodes within the graph and pass the streaming data on the edges between the nodes. In [26], the authors proposed a stream processing system, named Apache Storm, which is a distributed, reliable, and fault-tolerant SPS. Study [4] proposed a SPS, named Apache Flink that is used for computing unbounded and bounded streaming data using a stateful computing framework and a distributed processing engine. In [16], the authors proposed a SPS, named Twitter Heron that is a real-time, distributed, and fault-tolerant stream processing engine. An SPT is a DAG task written by users and running in a specific SPS.

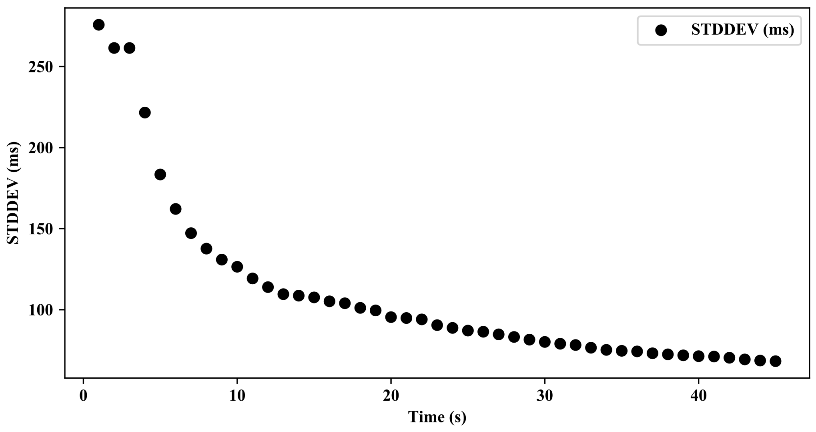
Ensemble learning: Study [5] first proposed the concept of ensemble learning. In [23], the authors used Boosting algorithm to combine multiple weak classifiers into a strong classifier. This algorithm makes ensemble learning to become an important research area. Study [9] proposed AdaBoost ensemble learning algorithm that is efficient and widely used in many fields. In [3], the authors proposed random forest algorithm that has achieved good results in many fields, so it is regarded as one of the best algorithms in machine learning. The authors proposed the integration of Gradient Boosting Decision Tree (GBDT) algorithm in [10]. GBDT is also a member of the Boosting family. In [12], the authors proposed the ExtraTree algorithm that can construct a completely randomized tree in extreme cases and its structure is independent of the output value of the learning sample.

Performance evaluation and prediction for SPSs: Most current performance evaluations for SPSs are based on experience methods [22] [24]. These methods first deployed SPTs in a specific SPS, and then collected performance metrics for task feedback, e.g., latency, throughput, etc. The performance prediction for SPSs is also like performance evaluation [28]. The above two types of performance evaluation and prediction research mainly focused on the impact of hardware resource on the performance.

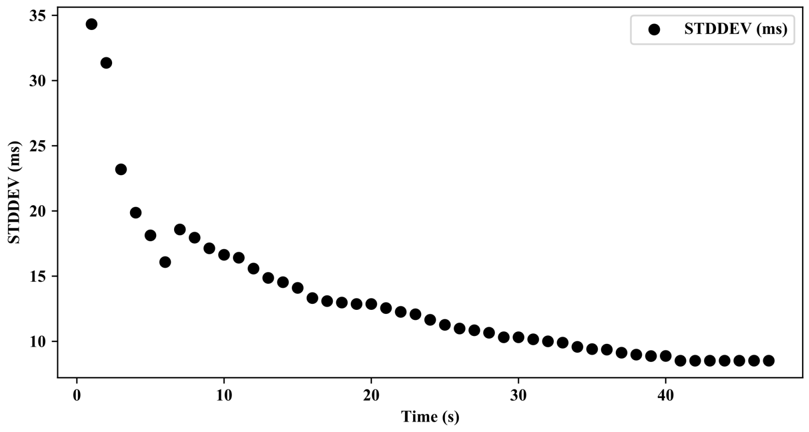
In an actual production environment, once a SPT is deployed, changes in the task will affect the execution of SPT. The best way is to predict performance in advance and prevent it from happening this situation. The most direct impact on the performance is the data rate of the SPT. In this paper, with the latency guarantees, the performance volatility of SPTs was analyzed and the ExtraTree algorithm was used to predict the throughput of SPTs under different data rates to avoid the situation of unavailable services.

3. Performance Volatility

In this section, we mainly describe the performance volatility of SPTs during execution. To obtain the performance of the task under a relatively stable state, a volatility detection algorithm was proposed and analyzed.



(a) Performance volatility on WordCount.



(b) Performance volatility on Iteration.

Fig. 1. Performance volatility on WordCount and Iteration.

3.1. Phenomenon of Performance Volatility

Performance volatility is a common phenomenon in which an SPT exhibits unstable performance over time under normal operating conditions, as shown in Fig.1 (data sets and experimental environment are described in Section 6).

The horizontal axis of Fig 1(a) represents the running time of an SPT, and the vertical axis represents the standard deviation of the latency metric. It can be clearly seen from the Fig 1(a) that the standard deviation reaches more than 250 in the initial stage. After 40 s, standard deviation stays around 20 and keeps relatively stable.

The results of Fig 1(b) are similar to that of Fig 1(a) and shows the phenomenon that the latency metric fluctuates over time. After 40 s, the standard deviation of the latency metric is kept at about 5. The performance of SPTs has the following characteristics: (1) the performance of SPTs fluctuates over time; (2) performance volatility tend to decline over time and eventually tend to be relatively stable.

3.2. Volatility Detection

Generally, the metrics for volatility detection of samples have extreme values, variances, and standard deviations. These metrics are formulated as follows:

$$X_{range} = \max(X) - \min(X) \tag{1}$$

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{X}\right)^2 \tag{2}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{X}\right)^2} \tag{3}$$

In Equation 1, the extreme value X_{range} is obtained by subtracting the minimum value from the maximum value in the sample set. In Equation 2, the variance metric σ^2 is the average $\bar{X}$ of the squared value of the difference between the average of each sample value x_i and the total sample size. In Equation 3, the standard deviation σ is the square root of the variance σ^2.

The extreme metric is very susceptible to noise from the samples. The variance metric is used to measure the volatility of a group of samples, that is, the deviation of a group of samples from the mean of samples. Similarly, the standard deviation can also reflect the degree of deviation among samples. However, the value of the variance is the square of the difference between the sample and the mean, which is greatly affected by the sample data. Therefore, it is more reasonable to use the standard deviation as a measure of the performance volatility for SPTs. The performance volatility detection algorithm uses the standard deviation to measure volatility.

By executing the volatility detection algorithm, the performance volatility values of SPTs are easily and efficiently calculated at a certain moment, but we do not know whether the volatility values are in a relatively stable state. In Section 4, we will evaluate stable states and output performance.

4. Performance Evaluation

In this section, we first perform a regression algorithm on the volatility values described in the previous section, and then elaborate on the choice of regression functions. Finally, the algorithm outputs the performance values in a relatively stable state, that is, evaluates the relative steady-state performance of SPTs.

4.1. Volatility Regression

To reduce the burden on humans to observe performance volatility, it is necessary to intelligently identify performance when an SPT is in a relatively stable state. Through the description of the performance volatility described in Section 3.1, the volatility will decrease and become relatively stable over time. To do this, we first perform a polynomial regression on the performance volatility, as shown in follows:

$$\hat{y}(w,\sigma) = w_0\sigma_0^0 + w_1\sigma_1^1 + \cdots + w_m\sigma_m^m + \xi(\sigma) \tag{4}$$

where w_i denotes the weight, and σ_i^i denotes the performance volatility value. If $\sigma_0 = 1$, Equation 4 is formulated as follows:

$$\hat{y}(w,\sigma) = \sigma \cdot W + \xi(\sigma) \tag{5}$$

where σ represents an $n \times (M+1)$ matrix, and W represents a $(M+1) \times 1$ matrix. In Equation 4 and 5, $\xi(\sigma)$ represents the error function and it is formulated as follows:

$$\xi(\sigma) = \min\left\{\|\sigma \cdot \mathrm{W} - y\|^2 + \alpha\|w\|^2\right\} \tag{6}$$

Regression task is performed by minimizing the sum of squared errors, and α is used for controlling the amount of expansion and contraction of the coefficients. Thus, the regression function of the performance volatility of SPTs is obtained by polynomial regression, and the derivative $\hat{y}'(w,\sigma)$ of the regression function was obtained. The problem of determining whether an SPT is in a relatively stable state is converted into a problem of calculating derivative value of regression function, i.e., $\hat{y}'(w,\sigma)$. However, this method will lead to another problem in selecting regression functions, because configuring different regression items will obtain different regression functions.

Some regression functions are capable of solving the volatility selection problem well, but others will bring unsatisfactory results. This phenomenon is shown in the experiment in Section 5. Ideally, $\hat{y}'(w,\sigma)$ close to 1 or -1 means that the performance of an SPT is more unstable, and close to 0 proves that the performance is stable.

4.2. Regression Selection

The R-squared value R^2, called the coefficient of determination, reflects the proportion of all variation of the dependent variable that can be interpreted by the independent variable through the regression relationship. The higher the value, the better the algorithm. The maximum value is 1, and R^2 is formulated as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{m}\left(\sigma_i - \hat{\sigma}\right)^2}{\sum_{i=1}^{m}\left(\sigma_i - \bar{\sigma}\right)^2} \tag{7}$$

By calculating the R^2 of the corresponding function of multiple regression terms, the function with the largest R^2 is selected. This function is the best choice among candidate functions, and the algorithm is briefly described in Algorithm 1.

Regarding the time complexity of Algorithm 1, the complexity of the loop in step 1 is $O(m)$, the loop in step 2 is $O(l)$, the loop in step 3 is $O(l)$, the loop in step 4 is $O(l)$.

```
Algorithm 1: RIS (Regression Item Selection)
Input: Performance volatility set $F = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$, regression item set
       $D = \{d_1, d_2, \ldots, d_l\}$.
Output: Regression function $\widehat{y}_{max}$.
begin
    (1) Calculate the mean of samples $\overline{\sigma}$ in $F$:
    for $i \leftarrow 0$ to m-1 do
        $\overline{\sigma} \leftarrow \overline{\sigma} + \sigma_i$;
    end
    $\overline{\sigma} \leftarrow \frac{\overline{\sigma}}{m}$;
    (2) Calculate regression set $Y = \{\widehat{y_1}, \widehat{y_2}, \ldots, \widehat{y_l}\}$ using regression item set
     $D = \{d_1, d_2, \ldots, d_l\}$:
    for $i \leftarrow 0$ to l-1 do
        $\widehat{y_i} \leftarrow Regression(d_i)$;
    end 3
    Calculate R-squared set $R^2 = \{r_1^2, r_2^2, \ldots, r_l^2\}$ using regression function set
     $Y = \{\widehat{y_1}, \widehat{y_2}, \ldots, \widehat{y_l}\}$ (refer Equation 7):
    for $i \leftarrow 0$ to l-1 do
        $R[i] \leftarrow 1 - \frac{\sum_{i=1}^{m}(\sigma_i - \hat{\sigma})^2}{\sum_{i=1}^{m}(\sigma_i - \bar{\sigma})^2}$;
    end 4
    Select the maximum $r_{max}^2$ in $R^2 = \{r_1^2, r_2^2, \ldots, r_l^2\}$:
    $r_{max}^2 \leftarrow 0$;
    for $i \leftarrow 0$ to l-1 do
        if $r_i^2 > r_{max}^2$ then
            $r_{max}^2 \leftarrow r_i^2$
        end
    end
    (5) Calculate the function $\widehat{y}_{max}$ in $Y$ using $r_{max}^2$:
    for $i \leftarrow 0$ to l-1 do
        if $r_i^2 = r_{max}^2$ then
            $\widehat{y}_{max} \leftarrow \widehat{y_i}$;
        end
    end
    return $\widehat{y}_{max}$;
end
```

In step 5, the loop is also $O(l)$. Therefore, the final time complexity of Algorithm 1 is $O(m + 4l)$.

Also, regarding the spatial complexity of Algorithm 1, the complexity of $\overline{\sigma}$ in step 1 is $O(1)$. The set Y in step 2 is $O(l)$. The loop R-square set in step 3 is $O(l)$, and r_{max}^2 in step 4 is $O(1)$. The $\widehat{y}_{maxl}$ in step 5 is $O(1)$. Therefore, the final spatial complexity of Algorithm 1 is $O(2l + 3)$, i.e., $O(l)$.

4.3. Performance Output under A Steady State

The optimal regression function is obtained through the regression term selection algorithm, i.e., Algorithm 1. When the value of the derivative function of the regression func-

Algorithm 2: POA (Performance Output Algorithm)

```
Input: Volatility regression function $\widehat{y}_{max}$, performance metric set
       $X = \{x_1, x_2, \ldots, x_n\}$.
Output: Performance metric $x_i$.
begin
    (1) Calculate the derivative $\widehat{y}^{'}_{max}$ using $\widehat{y}_{max}$;
    (2) Calculate each derivative $\widehat{y}^{'}_{max}(x_i)$ in $X = \{x_1, x_2, \ldots, x_n\}$:
    $Y[i] \leftarrow Null$;
    for $i \leftarrow 0$ to n-1 do
        $Y[i] \leftarrow \widehat{y}^{'}_{max}(x_i)$;
    end 3
    Output performance $x_i$ when the derivative value is equal to 0:
    for $i \leftarrow 0$ to l-1 do
        if $Y[i] == 0$ then
            return $x_i$;
        end
    end
end
```

tion is 0, an SPT enters a relatively stable state. At this time, the performance value is output through the performance output algorithm, i.e., Algorithm 2.

Algorithm 2 is relatively simple, so no specific analysis is performed here. The time complexity of the algorithm is $O(n)$, and the spatial complexity is $O(1)$.

5. Performance Prediction

In Section 4, the performance output algorithm outputs the throughput of an SPT in a relatively stable state, so the ExtraTree algorithm is used for predicting performance at different data rates. In this section, this algorithm is described.

5.1. ExtraTree Introduction

ExtraTree is a novel tree-based ensemble learning algorithm for supervising classification and regression problems. It mainly emphasizes on randomness and selection for segment point when splitting tree nodes. In extreme cases, it is constructed completely randomly. The structure of the tree is independent of the output values of the learning samples. Compared with the random forest algorithm, this algorithm has higher computational efficiency and higher accuracy. The ExtraTree algorithm is very similar to the random forest algorithm. Although they are composed of multiple decision trees, the ExtraTree and the random forest have two differences: (1) the random forest uses the Bagging algorithm, that is to say, the training samples for each weak learner are not all, but the ExtraTree uses all training samples to train every weak learner. In addition, ExtraTree adopts a random selection strategy to select features, so its results are better than random forests; (2) the random forest obtains the best bifurcation attribute in a random subset, but the ExtraTree obtains the bifurcation value completely and randomly to implement the bifurcation

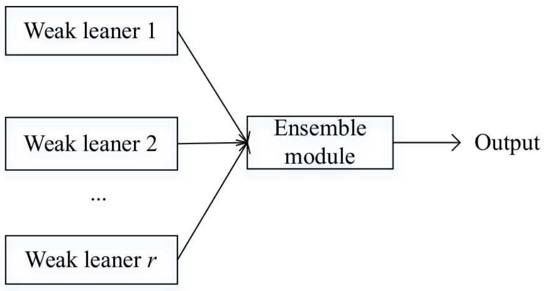


Fig. 2. The structure of ensemble learning.

of the decision tree. Ensemble learning forms a strong ensemble learning algorithm by constructing and combining multiple weak learners to complete specific learning tasks.

Fig 2 shows a general structure of ensemble learning that combines a group of weak learners through a specific strategy. Weak learners are usually trained by existing learning algorithms, such as C4.5 decision tree algorithm, BP neural network algorithm, etc. One of the most important advantages of ensemble learning is that the algorithm achieves superior excellent generalization than a single learner by combining multiple weak learners. In general, it combines non-optimal learners into one piece and gets the best learner. Therefore, the combination strategy for weak learners is particularly important. Assuming that ensemble learning includes T weak learners $h_1, h_1, \ldots, h_T$, where the output of h_i on x is $h_i(x)$. Average, and voting strategy are formulated as follows:

$$H(x) = \frac{1}{T}\sum_{i=1}^{T} h_i(x) \tag{8}$$

$$H(x) = \frac{1}{T}\sum_{i=1}^{T} w_i h_i(x) \tag{9}$$

where w_i is the weight of the weak learner h_i. To be noted, $w_i \geq 0$ and $\sum_{i=1}^{T} w_i = 1$ are required.

5.2. Algorithm Construction

Fig 3 shows a schematic diagram of the ExtraTree structure. The ExtraTree algorithm contains multiple decision trees, each of which contains a tree-like decision node sequence. Based on this sequence, the tree splits into various branches until it reaches the end of the tree (the leaf node). The prediction result of each decision tree is output through the leaf nodes, and the final outputs of the multiple decision trees are combined for prediction.

For the throughput prediction algorithm of SPTs, assuming that data set is $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where N denotes the sample size, x_i denotes the sample data, and y_N denotes the throughput of an SPT. When generating each decision

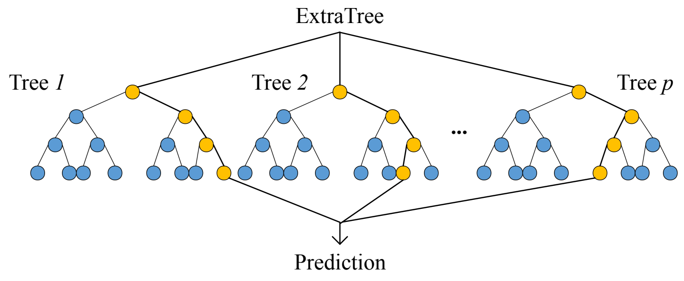


Fig. 3. The structure of ExtraTree.

Algorithm 3: EBA (ExtraTree Building Algorithm)

```
Input: Data set $D$, the number of trees $N_t$.
Output: ExtraTree $F_{tree}$.
begin
    for $i \leftarrow 0$ to $N_t - 1$ do
        (1) Calculate the optimal feature $j$ and the point $s$ to split current node:
        $\min_{j,s}[\sum_{x_i \in R_1(j,s)} (y_i - \hat{c}_1)^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{c}_2)^2]$;
        (2) Calculate output value $\widehat{c_m}$ using $min(j, s)$ in the current node:
        $\widehat{c_m} \leftarrow \frac{1}{N_m} \sum_{x_i \in R_m(j,s)} y_i$,
        where $R_1(j,s) = \left\{x|x^{(j)} \leq s\right\}, R_2(j,s) = \left\{x|x^{(j)} > s\right\}$;
        (3) Repeat (1) and (2) using $R_1(j,s)$ and $R_2(j,s)$;
        (4) Divide input space into $m$ nodes $R_1, R_2, \ldots, R_m$ and generate decision tree
         $f_i(x)$:
        $f_i(x) \leftarrow \sum_{m=1}^{M} \hat{c}_m I\,(x \in R_m)$.
        where $I = \begin{cases} 1\ if\ (x \in R_m) \\ 0\ if\ (x \notin R_m) \end{cases}$;
        (5) Add current decision tree $f_i(x)$ into $F_{tree}$:
        $F_{tree}[i] \leftarrow f_i(x)$;
    end
    return $F_{tree}$
end
```

tree, the algorithm calculates the best features j and output value s, as shown in follows:

$$\min_{j,s} \left[\sum_{x_i \in R_1(j,s)} (y_i - \hat{c}_1)^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{c}_2)^2 \right] \tag{10}$$

where $R_1(j,s) = \left\{x|x^{(j)} \leq s\right\}$ and $R_2(j,s) = \left\{x|x^{(j)} > s\right\}$ are two regions divided by j and s. $\hat{c}_1$ and $\hat{c}_2$ are the throughput output values. In addition, $(y_i - \hat{c}_1)^2$ and $(y_i - \hat{c}_2)^2$ are the mean square error (MSE). The algorithm repeats the above steps until all features are segmented, and the construction process is shown in Algorithm 3.

For algorithm 3, it is assumed that the number of features is k, steps 1-2 need to be repeated k times. Moreover, a total of cycles is required N_t. Therefore, the time complexity of Algorithm 3 is $O(kN_t)$.

Algorithm 4: TPA (Throughput Prediction Algorithm)

```
Input: ExtraTree $F_{tree}$, Sample $x$.
Output: The predicted throughput $p$.
begin
    $p_{sum} \leftarrow 0$;
    for $i \leftarrow 0$ to $N_t - 1$ do
        Predict throughput $p$ and add it to $p_{sum}$:
        $p_{sum} \leftarrow p_{sum} + f_i(x)$;
    end
    Calculate the mean predicted throughput:
    $p \leftarrow \frac{p_{sum}}{N_t}$;
    return $p_{sum}$;
end
```

5.3. Throughput Prediction

The output of Algorithm 3 during the prediction phase is the mean of the output values of multiple decision trees and it is formulated as follows:

$$f(x) = \frac{1}{N_t} \sum_{i=1}^{N_t} f_i(x) \tag{11}$$

where N_t is the number of decision trees, $f_i(x)$ is the predicted throughput value, and the prediction process is shown in Algorithm 4.

Algorithm 4 is relatively simple, giving the time complexity of the algorithm is $O(N_t)$, and the space complexity is $O(1)$.

6. Experimental Evaluation

In this section, we describe the methodology, experimental environment, evaluation metrics, volatility regression, comparison of errors, comparison of accuracy and efficiency, and the impact of different sample ratio on errors in detail.

6.1. Methodology

In the experiments, the proposed methodology and the proposed prediction model are applied on an evolving data stream. The overall work principal is shown in Fig. 4.

As shown in Fig. 4, the experimental methodology consists of two components: (1) Online prediction, and (2) Offline learning. The first component firstly detects volatility (Section 3), and then evaluates performance (Section 4) in a real-time fashion. When the performance in steady state is evaluated, the output performance is used to predict the throughput in a real-time style. In addition, an copy of the output is used for training the proposed model in a offline style. During the offline learning phase, the model continuously optimizes itself.

Two open-source benchmarks, i.e., WordCount (WC), and Iteration (ITE) were used to evaluate the effectiveness of the proposed method. An external server was built outside

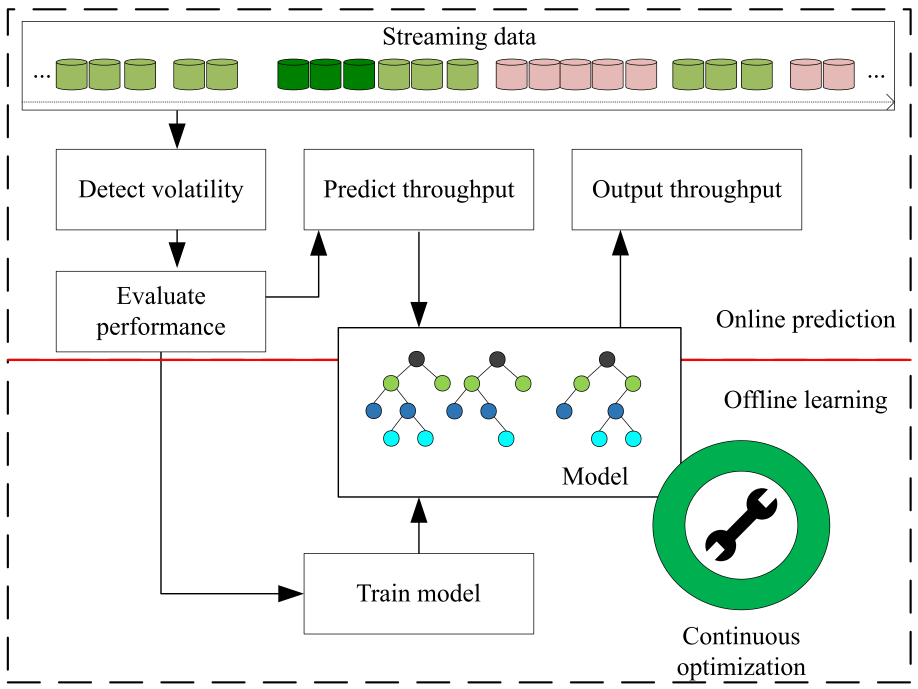


Fig. 4. The experimental methodology.

an SPS cluster that includes one JobManager and three TaskManagers. The external server undertaken the task of collecting throughout of SPTs in real time, and executed real-time throughput prediction for SPTs.

In these benchmarks, WC sent English sentences to an SPT by configuring different sending rates. The SPT first segmented the received English sentences, and then continuously counted the number of occurrences of each word. ITE continuously sent values to an SPT, and then the SPT iteratively calculated the values. Table 1 summarized all data sets from two benchmarks.

Table 1. The description of data sets.

Benchmarks	Total sample size	Training sample size	Predicting sample size
WC	99,980	79,984	19,996
ITE	100,002	80,002	20,000

During the performance prediction phase, three ensemble learning algorithms were used to compare the proposed algorithm.

AdaBoost (Adaptive Boosting), a typical Boosting algorithm, belongs to the Boosting algorithm family. The core of the algorithm is the process of promoting weak learner to a strong learner. The working mechanism is as follows: (1) a weak learner is trained from the initial training set; (2) the training sample distribution is adjusted according to the performance of the weak learner, so that the training samples of the previous weak learner's errors receive more attention on the subsequent training process; (3) train the next weak learner based on the adjusted sample distribution. Repeat these processes until the number of weak learners reaches T, and finally combine the weights of all weak learners. The AdaBoost algorithm is a linear combination based on the weak learners, and it is formulated as follows:

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{12}$$

where α_t is the proportion of weak learners to a strong learner, which is different from the weighted average method.

GBDT is also a member of the Boosting family. When training a single weak learner, the algorithm considers the loss function of the previous weak learners. In addition, GBDT also uses an iterative approach through a forward-distributed algorithm. Note that weak learners in this algorithm can only use the CART regression tree algorithm.

Random Forest (RF), an extension of Bagging, is based on the ensemble learning of Bagging with decision tree learners, and adds the characteristics of random attribute selection. The Bagging randomly selects training data, and then constructs multiple weak learners. Finally, it combines multiple decision trees to improve the overall performance. In short, a random forest is obtained by constructing multiple decision trees and merging all the decision trees together to achieve accurate and stable prediction results. It has the advantages of simplicity, easy implementation, and low computational cost. Therefore, this algorithm is one of the comparison algorithms in this paper.

Additionally, to fairly evaluate the performance of different algorithms, the parameter values for each algorithm used in the experimental study is set to the same. Main parameter configurations are summarized in Table 2.

Table 2. Main parameter configurations of different algorithms.

Parameters	Values	Description
n_estimators	50	The number of trees in the forest.
max_depth	30	The maximum depth of the tree.
min_samples_split	2	The minimum number of samples required to split an internal node.
min_samples_leaf	1	The minimum number of samples required to be at a leaf node.

Other parameter configurations use the default values in scikit-learn packages [19].

6.2. Experimental Environment

There are many popular SPSs, such as Apache Flink, Twitter Heron, Apache Storm, Apache Spark, etc. Apache Spark simulates real-world stream processing using a micro-batch processing approach. Twitter Heron is an enhanced version of Apache Storm. Because Apache Flink has strong state support and high performance for streaming data, it was used as the carrier for all experiments.

Apache Flink is built in a local cluster consisting of four servers. One server is the JobManager (Master) and others are TaskManagers (Slaves). The JobManager server is mainly responsible for the distribution and coordination of tasks, and TaskManager servers are mainly responsible for executing specific SPTs, i.e., WC, and ITE. The four servers in the cluster have the same hardware configuration as the external server. CPU is "Intel(R) Core(TM) i7-4790 CPU 3.60GHz", memory is 8 GB, hard disk is 500 G, and operating system is CentOS-6.5.

6.3. Evaluation Metrics

To evaluate the performance of the proposed method for the throughput prediction of SPTs, six metrics were used to compare different algorithms.

(1) Explain variance (EV) is a measure of the ability of a regression equation to explain the degree of change in the dependent variable or the degree to which the equation fits a sample. The closer the EV is to 1, the better the algorithm, and the lower the value, the worse the algorithm. EV is formulated as follows:

$$EV = 1 - \frac{\operatorname{Var}(y_i - \hat{y})}{\operatorname{Var}(y_i)} \tag{13}$$

(2) The R-squared value (R^2) is the degree to which the regression equation characterizes the dependent variables. The R-squared of the best algorithm is 1, and the difference is smaller. This metric is formulated as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{m} (y_i - \hat{y})^2}{\sum_{i=1}^{m} (y_i - \bar{y})^2} \tag{14}$$

(3) The mean absolute error (MAE) is the average difference between the predicted value and the true value, and it is formulated as follows:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |f_i - y_i| = \frac{1}{n} \sum_{i=1}^{n} |e_i| \tag{15}$$

(4) The mean square error (MSE) is the expected value of the square of the difference between the predicted value and the true value. It is recorded as a convenient method to measure the average error. It was used to evaluate how much the data has changed. The smaller the value, the better the prediction algorithm will describe the data in the experiments. This metric formulated as follows:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (f_i - y_i)^2 \tag{16}$$

(5) The root mean square error ($RMSE$) is the arithmetic square root of the mean square error, and it is formulated as follows:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (f_i - y_i)^2} = \sqrt{MSE} \tag{17}$$

(6) The median absolute error ($MediaAE$) is formulated as follows:

$$MediaAE = \text{media}\left(\|y_i - \hat{y}_i|, \ldots, |y_n - \hat{y}_n|\right) \quad (18)$$

From Equation 13 to Equation 18, f_i is the predicted throughput value, y_i is the real throughput value, $e_i = |f_i - y_i|$ is the absolute error. n is the sample size, $\overline{y}$ is the mean of throughput.

6.4. Volatility Regression

In Section 3, polynomial regression was introduced and performed on performance volatility values. In this experiment, by configuring different regression terms $D = \{d_1, d_2, \cdots, d_l\}$, Equation 3 was used to calculate the corresponding regression function set. Regression items were set to 2, 3, 4, and 5, respectively. The experimental results were shown in 5.

In Fig 5, there was a clear trend of increasing prediction performance as terms increased. For example, the regression function and the performance volatility values were very different when the regression term was set to 2. In contrast, when the regression term was set to 5, the function could well regress the performance volatility values. The results of Fig 5 showed that the tangent of regression function on WC was -1 when the time was about 30 s, but the corresponding performance volatility scatter plot did not reach a relative stable state, which explained the necessity of regression term selection, i.e., Algorithm 1, in Section 4. The results from other benchmarks, i.e., ITE, showed the similar trend as shown in Fig 5. If the algorithm output the throughput of an SPT at this time, meaning that it was not in a relatively stable performance. When the regression term became larger, e.g., 5, the output of performance was more representative of the throughput in a relatively stable state, as shown in Fig 5(d), and Fig 5(h). To evaluate the performance of regression item selection algorithm in more detail, the R-squared values R^2 of different regression items were shown in Table 3.

Table 3. R^2 of different regression items.

Benchmarks	Item-2	Item-3	Item-4	Item-5
WC	0.87	0.95	0.98	0.98
ITE	0.84	0.90	0.93	0.96

In Table 3, R^2 gradually approached 1 as the regression term on WC, and ITE increased, which also indicated that the selection of regression terms was necessary in the performance volatility regression process. Therefore, the regression algorithm based on excellent performance was more accurate to judge a relatively stable state, and the output was more reasonable.

6.5. Comparison of errors

The purpose of this experiment was to compare the prediction errors of different ensemble learning algorithms under a steady state at different data rates. The experimental results were shown in Fig 6.

Table 4. The errors of different ensemble learning algorithms.

Benchmarks	MAE				MSE				MediaAE			
	A	E	G	R	A	E	G	R	A	E	G	R
WC	0.21	**0.15**	0.19	0.23	0.15	**0.04**	0.14	0.08	0.09	**0.07**	0.12	0.13
ITE	0.12	**0.05**	0.10	0.09	0.09	**0.03**	0.10	0.12	0.05	**0.02**	0.13	0.10
AVG	0.165	**0.10**	0.145	0.16	0.12	**0.035**	0.12	0.10	0.07	**0.045**	0.125	0.115

Note: (A) AdaBoost; (E) ExtraTree; (G) GBDT; (R) Random Forest; (AVG) Average.

As shown in Fig 6, there were differences between the four ensemble learning algorithms, and ExtraTree had lower errors compared with other algorithms, i.e., AdaBoost, GBDT, and Random Forest. Table 4 summarized the detailed results.

As shown in Table 4, MAE, MSE, and $MediaAE$ of ExtraTree on all benchmarks were lower than that of other algorithms. For instance, MAE of ExtraTree, AdaBoost, GBDT, and Random Forest on the benchmark WC were 0.15, 0.21, 0.19, and 0.23, respectively, which showed that the ExtraTree had the lowest error. On the benchmark ITE, MSE and $MediaAE$ of all algorithms showed the same results. Moreover, the average of MAE, MSE, and $MediaAE$ of the ExtraTree were 0.10, 0.035, 0.045, respectively, which indicated the ExtraTree had the lowest errors compared with other algorithms.

6.6. Comparison of Accuracy and Efficiency

The purpose of this experiment was to compare the accuracy and efficiency of different ensemble learning algorithms for the throughput prediction of SPTs. Accuracy and execution time are formulated as follows:

$$Accuracy = \left(100 - \frac{1}{n}\sum_{i=1}^{n}\left|\frac{f_i - y_i}{y_i}\right|\right) \times 100\% \tag{19}$$

where, n is the number of samples, f_i is the predicted throughput value, y_i is the actual throughput value.

$$ET = \frac{10000}{n}\sum_{i=1}^{n} t_i \tag{20}$$

where, n is also the number of samples, and t_i is the prediction execution time for one sample. Since the execution time to predict one sample is short and not good for comparison, the constant coefficient 10,000 in Equation 20 is used to estimate the prediction time for per 10,000 samples. Table 5 summarized the experimental results from the benchmark WC, and ITE.

From Table 5, it was apparent that ExtraTree on all the benchmarks resulted in the highest values of accuracy and efficiency. For example, the accuracy of ExtraTree on all the benchmarks had the highest rates with 91.13%, and 89.94%. The ExtraTree had the lowest execution time with 0.82, and 0.85 s/10,000 samples. These results indicated that the ExtraTree had the highest accuracy and the highest efficiency compared with other algorithms on all benchmarks.

Table 5. Accuracy and efficiency of different algorithms.

Benchmarks	Accuracy (%)				ET (s/10,000 samples)			
	A	E	G	R	A	E	G	R
WC	68.36	**91.13**	75.59	67.84	0.96	**0.82**	0.85	0.82
ITE	70.61	**89.94**	77.55	69.71	0.95	**0.85**	0.86	0.86
AVG	69.485	**90.535**	76.57	68.775	0.955	**0.835**	0.855	0.84

Note: (A) AdaBoost; (E) ExtraTree; (G) GBDT; (R) Random Forest; (AVG) Average.

6.7. The Impact of Different Sample Ratio on Errors

To verify the generalization ability of the proposed method, the prediction errors of different training sample ratios were used. The experimental results were shown in Fig 7.

As shown in Fig 7, as the proportion of training samples increased, EV of all ensemble learning algorithms gradually approached 1, and MAE, MSE and $MediaAE$ also gradually approached zero. In addition, all ensemble learning algorithms was stable in terms of EV. Moreover, all algorithms showed a clear trend of decreasing errors. These results indicated that the generalization ability of ExtraTree was stable, and it had lower errors compared with other ensemble learning algorithms.

7. Conclusion

In this paper, the performance volatility of SPTs were studied and the corresponding volatility detection algorithm was proposed to accurately output the throughput of SPTs under a relatively stable state. In the throughput prediction phase, an ExtraTree-based algorithm was used for predicting the throughput. In the experiments, the prediction performance (i.e., errors, accuracy, and efficiency) of different ensemble learning algorithms on two benchmarks were compared. The results illustrated that the proposed algorithm had low error rates and high accuracy rates with a relatively high efficiency. Based on the research results, our future work will focus on performance prediction in heterogeneous environments, which requires a deeper study of the internal details of SPTs.

Acknowledgments. The authors are grateful to the anonymous referees for their insightful suggestions and comments. This research was supported by the National Natural Science Foundation of China under Grant Nos. 61862060, 61462079, 61562086 and 61562078, and the Doctoral Science and Technology Innovation Project in Xinjiang University under Grant No. XJUBSCX-201901.

References

1. Abe, G., Sato, K., Itoh, M.: Driver trust in automated driving systems: The case of overtaking and passing. IEEE Transactions on Human-Machine Systems 48(1), 85–94 (2017)
2. Borthakur, D.: The hadoop distributed file system: Architecture and design. Hadoop Project Website 11(2007), 21 (2007)
3. Breiman, L.: Random forests. Machine learning 45(1), 5–32 (2001)

4. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache flink: Stream and batch processing in a single engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 36(4) (2015)
5. Dasarathy, B.V., Sheela, B.V.: A composite classifier system design: concepts and methodology. Proceedings of the IEEE 67(5), 708–713 (1979)
6. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
7. Edwards, L.: Public relations, voice and recognition: a case study. Media, Culture & Society 40(3), 317–332 (2018)
8. Etter, M., Ravasi, D., Colleoni, E.: Social media and the formation of organizational reputation. Academy of Management Review 44(1), 28–52 (2019)
9. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences 55(1), 119–139 (1997)
10. Friedman, J.H.: Stochastic gradient boosting. Computational statistics & data analysis 38(4), 367–378 (2002)
11. Gedik, B., Schneider, S., Hirzel, M., Wu, K.L.: Elastic scaling for data stream processing. IEEE Transactions on Parallel and Distributed Systems 25(6), 1447–1463 (2013)
12. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine learning 63(1), 3–42 (2006)
13. Ghajargar, M., Wiberg, M., Stolterman, E.: Designing iot systems that support reflective thinking: A relational approach. International Journal of Design 12(1), 21–35 (2018)
14. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future generation computer systems 29(7), 1645–1660 (2013)
15. Handel, T., Schreiber, M., Rothmaler, K., Ivanova, G.: Data security and raw data access of contemporary mobile sensor devices. In: World Congress on Medical Physics and Biomedical Engineering 2018. pp. 397–400. Springer (2019)
16. Kulkarni, S., Bhagat, N., Fu, M., Kedigehalli, V., Kellogg, C., Mittal, S., Patel, J.M., Ramasamy, K., Taneja, S.: Twitter heron: Stream processing at scale. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 239–250. ACM (2015)
17. Moertini, V.S., Suarjana, G.W., Venica, L., Karya, G.: Big data reduction technique using parallel hierarchical agglomerative clustering. IAENG International Journal of Computer Science 45(1), 188 – 205 (2018)
18. Nasir, M.A.U., Morales, G.D.F., Garcia-Soriano, D., Kourtellis, N., Serafini, M.: The power of both choices: Practical load balancing for distributed stream processing engines. In: 2015 IEEE 31st International Conference on Data Engineering. pp. 137–148. IEEE (2015)
19. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. Journal of Machine Learning Research 12, 2825–2830 (2011)
20. Rossi, A., Vila, Y., Lusiani, F., Barsotti, L., Sani, L., Ceccarelli, P., Lanzetta, M.: Embedded smart sensor device in construction site machinery. Computers in Industry 108, 12–20 (2019)
21. Rossion, B.: Humans are visual experts at unfamiliar face recognition. Trends in cognitive sciences 22(6), 471–472 (2018)
22. Samosir, J., Indrawan-Santiago, M., Haghighi, P.D.: An evaluation of data stream processing systems for data driven applications. Procedia Computer Science 80, 439–449 (2016)
23. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Machine learning 37(3), 297–336 (1999)
24. Sun, D., Yan, H., Gao, S., Zhou, Z.: Performance evaluation and analysis of multiple scenarios of big data stream computing on storm platform. KSII Transactions on Internet & Information Systems 12(7) (2018)

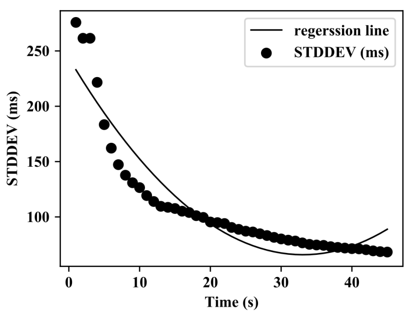
25. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment 2(2), 1626–1629 (2009)
26. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J.M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et al.: Storm@ twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data. pp. 147–156. ACM (2014)
27. Vora, M.N.: Hadoop-hbase for large-scale data. In: Proceedings of 2011 International Conference on Computer Science and Network Technology. vol. 1, pp. 601–605. IEEE (2011)
28. Wang, K., Khan, M.M.H.: Performance prediction for apache spark platform. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems. pp. 166–173. IEEE (2015)
29. Xu, J., Chen, Z., Tang, J., Su, S.: T-storm: Traffic-aware online scheduling in storm. In: 2014 IEEE 34th International Conference on Distributed Computing Systems. pp. 535–544. IEEE (2014)
30. Young, T., Hazarika, D., Poria, S., Cambria, E.: Recent trends in deep learning based natural language processing. ieee Computational intelligenCe magazine 13(3), 55–75 (2018)

Zheng Chu born in 1991. Ph.D. candidate in the School of Information Science and Engineering, Xinjiang University. His main research interests include distributed computing, in-memory computing, and machine learning.

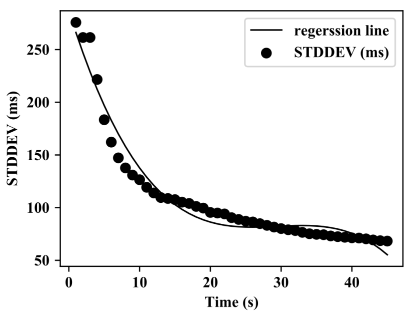
Jiong Yu born in 1964. Professor and Ph.D. supervisor in the School of Information Science and Engineering, Xinjiang University. His main research interests include grid computing, parallel computing, etc.

Askar Hamdulla born in 1972. Professor and PhD supervisor in the School of Information Science and Engineering, Xinjiang University. His main research interest is natural language processing.

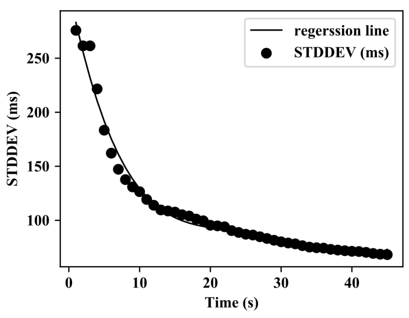
Received: January 31, 2020; Accepted: July 15, 2020.



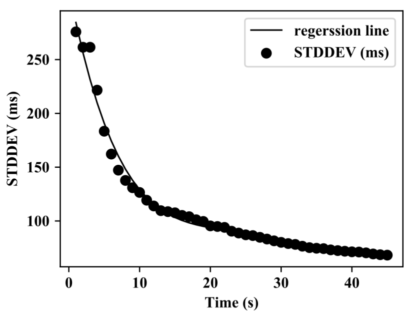
(a) WC (Item=2).



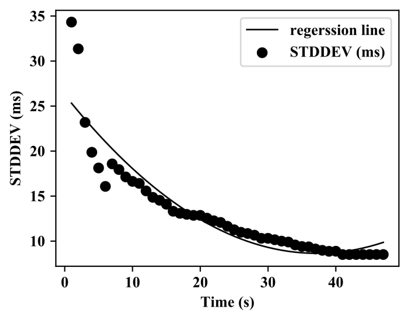
(b) WC (Item=3).



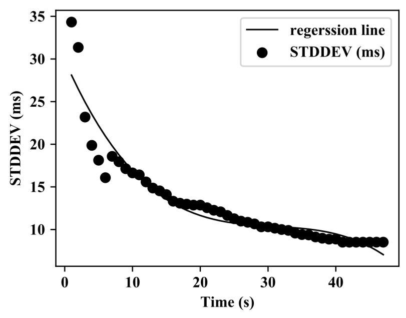
(c) WC (Item=4).



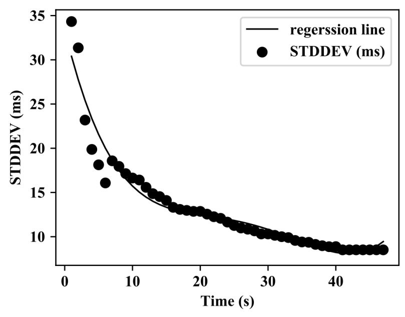
(d) WC (Item=5).



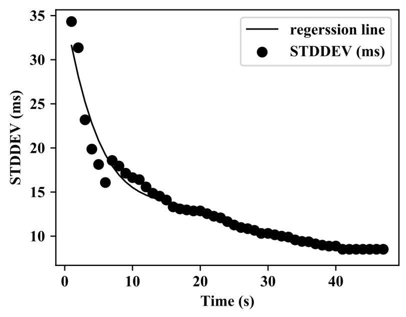
(e) ITE (Item=2).



(f) ITE (Item=3).

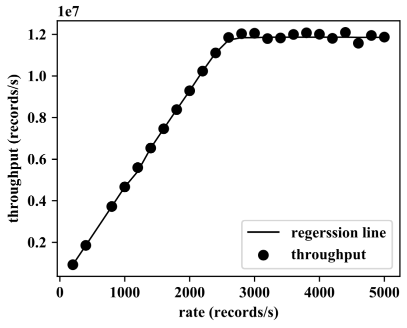


(g) ITE (Item=4).

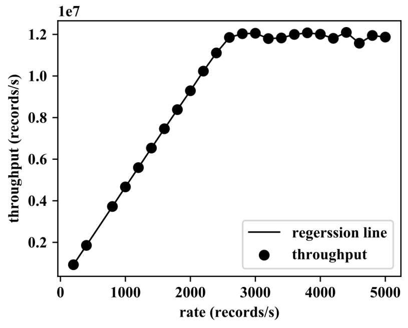


(h) ITE (Item=5).

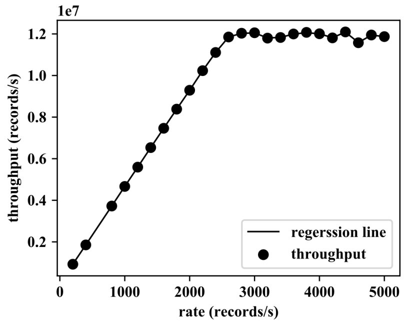
Fig. 5. Performance volatility regression using different regression Item. Note that STDDEV denotes the standard deviation of performance volatility.



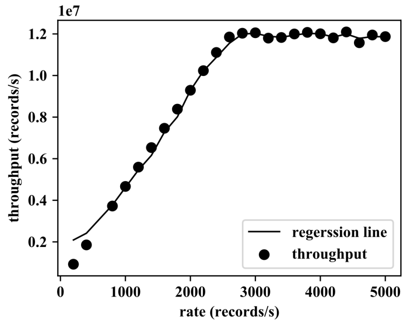
(a) AdaBoost (WC).



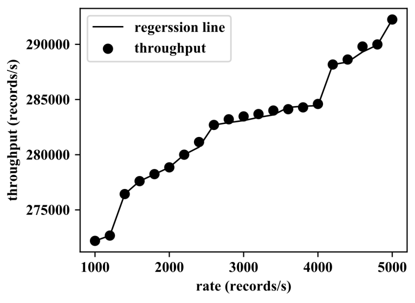
(b) ExtraTree (WC).



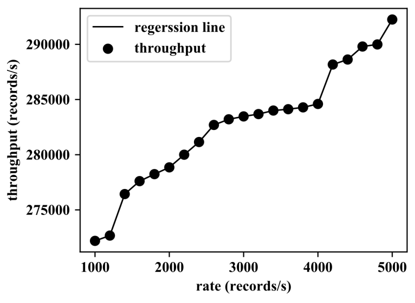
(c) GBDT (WC).



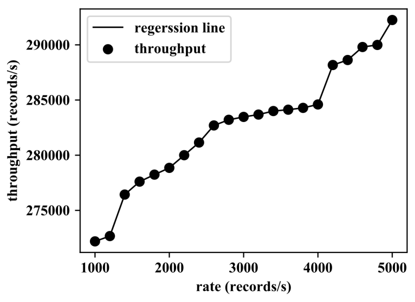
(d) Random Forest (WC).



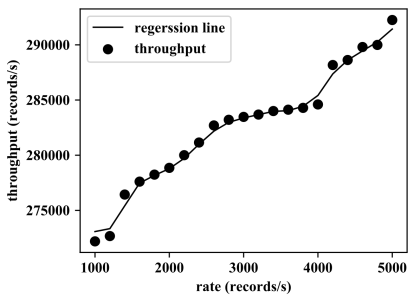
(e) AdaBoost (ITE).



(f) ExtraTree (ITE).

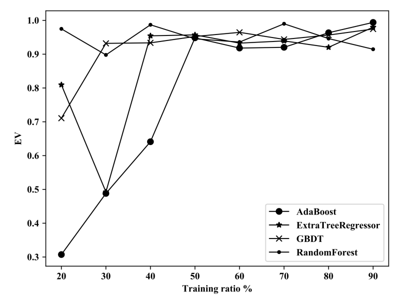


(g) GBDT (ITE).

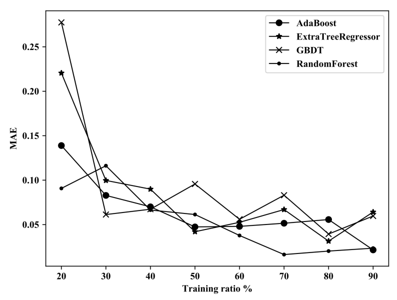


(h) Random Forest (ITE).

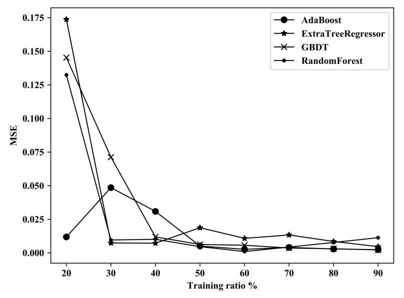
Fig. 6. Throughput regression using different ensemble learning algorithms.



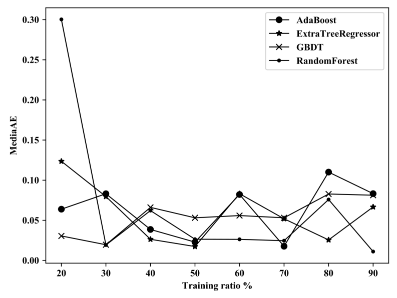
(a) EV (WC).



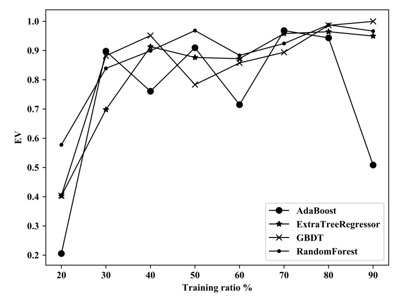
(b) MAE (WC).



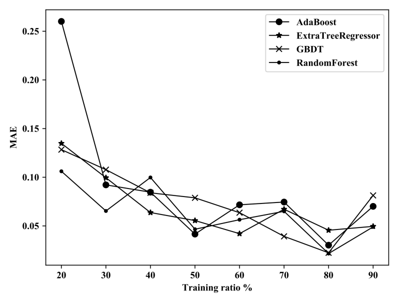
(c) MSE (WC).



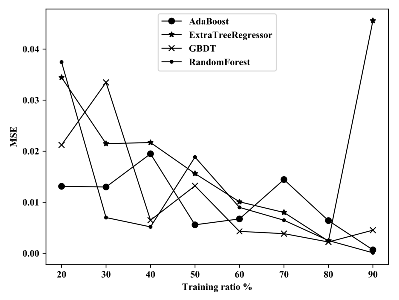
(d) $MediaAE$ (WC).



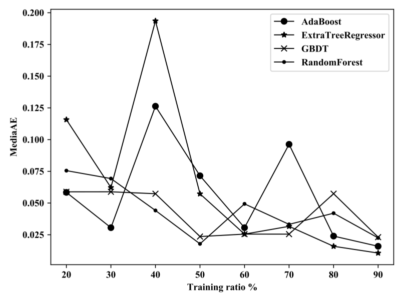
(e) EV (ITE).



(f) MAE (ITE).



(g) MSE (ITE).



(h) $MediaAE$ (ITE).

Fig. 7. Errors using different ensemble learning algorithms at different training data ratio.