Big Data Research ••• (••••) •••–•••

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

An Efficient Time Optimized Scheme for Progressive Analytics in Big Data

Kostas Kolomvatsos [a], Christos Anagnostopoulos [b], Stathes Hadjiefthymiades [c]

[a] Department of Computer Science, University of Thessaly, 35100, Greece
[b] School of Computing Science, University of Glasgow, G12 8QQ, UK
[c] Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, 15784, Greece

ARTICLE INFO

Article history:
Received 11 June 2014
Received in revised form 3 December 2014
Accepted 3 February 2015
Available online xxxx

Keywords:
Big data
Continuous queries
Progressive analytics
Sequential time-optimized models

ABSTRACT

Big data analytics is the key research subject for future data driven decision making applications. Due to the large amount of data, progressive analytics could provide an efficient way for querying big data clusters. Each cluster contains only a piece of the examined data. Continuous queries over these data sources require intelligent mechanism to result the final outcome (query response) in the minimum time with the maximum performance. A *Query Controller* (*QC*) is responsible to manage continuous/sequential queries and return the final outcome to users or applications. In this paper, we propose a mechanism that can be adopted by the *QC* capable of managing partial results retrieved by a number of processors each one responsible for each cluster. Each processor executes a query over a specific cluster of data. The proposed mechanism adopts two sequential decision making models for handling the incoming partial results. The first model is based on a finite horizon time-optimized model and the second one is based on an infinite horizon optimally scheduled model. We provide mathematical formulations for solving the discussed problem and present simulation results. Through a large number of experiments, we reveal the advantages of the proposed models and give numerical results comparing them with a deterministic model. These results indicate that the proposed models can efficiently reduce the required time for returning the final outcome to the user/application while keeping the quality of the aggregated result at high levels.

1. Introduction

1.1. Motivation

Big data is an interesting research area and consists of the basis of future data driven decision making techniques. The appropriate management of huge amounts of structured or unstructured data is the key research challenge. This research area has attracted the attention of many institutes and companies worldwide. The reason is that, in many applications domains, huge amount of data are produced and stored requiring the appropriate management mechanisms and the so-called big data analytics. The increase of users' devices leads to an increased amount of data as well as to an increased number of queries. Decision makers should adopt analytics in order to reach efficient decisions according to the application domain. For instance, companies adopt analytics in order to deal with critical applications like security, customers behaviour, etc.

Handling big data is not a trivial issue. Many frameworks have been proposed so far. The most known framework is Hadoop[1] and its programming model MapReduce [11]. The idea behind Hadoop is to store data sets across distributed clusters and, then, run a distributed processing scheme in each cluster. In general, there are two processing models: (a) the **batch oriented** processing and, (b) the **stream (on-line) oriented** processing. Hadoop and MapReduce are oriented to batch processing. Problems arise when applications require real-time management of data streams. Specific extensions have already been proposed for Hadoop like Pig[2] and Hive[3] to handle these problems.

When data are 'static', batch processing is the most appropriate technique to retrieve analytics and build decision making mechanisms. For instance, a large company may want to discover patterns of the behaviour of buyers. It could adopt historical data to retrieve the discussed patterns and take specific decisions for the

E-mail addresses: kostasks@di.uoa.gr, kolomvatsos@cs.uth.gr (K. Kolomvatsos).

http://dx.doi.org/10.1016/j.bdr.2015.02.001
2214-5796/

[1] http://hadoop.apache.org/.
[2] http://pig.apache.org/.
[3] http://hive.apache.org/.

future strategy. In these cases, real-time data stream processing is not the appropriate technique. Consider now, a power plant, where security issues are very critical. Sensors for natural phenomena (e.g., temperature, people movement) result large amounts of data that should be managed in real-time. The aim is to create alarms when specific criteria are met and decisions should be taken related to the response to security violations. Real-time and big data analytics on streams require *new* models for handling the speed with which data arrive, are managed and stored.

Additionally, in the discussed application domain, there is the need of continuous query processing. Research efforts in streams query processing [1,4,6,10,17,26,34] focus on the ability of handling incoming data on-line against a set of continuous queries [25]. In this paper, we propose a mechanism that deals with continuous query processing for applications requiring the management of big data streams. *Progressive analytics* could constitute the basis of an efficient solution. Progressive analytics is the generation of early results to queries based on partial data, and the progressive refinement of these results as more data are received [5]. Progressive analytics requires only a few resources as query processing ends when sufficient accuracy is observed in early results. Furthermore, in order to avoid the users intervention in the decision on when to end the process, we propose the use of *Optimal Stopping Theory* (OST) [28]. We provide an intelligent mechanism that stops/terminates the process of collecting partial results and return the final outcome to the user/application. The proposed mechanism relies between the user/application and the progressive analytics service. The progressive analytics service is applied onto a number of clusters. A specific processor is responsible to (i) execute the same query in each cluster and (ii) return early (partial) results along with a confidence interval on these results to our mechanism. Based on the confidence values, the mechanism decides when it is the appropriate time to stop the process, alleviating users from monitoring the discussed information. Hence, the system saves time and resources in a setting of continuous queries over big data streams. The proposed mechanism is simple, however, efficient as we adopt the advantages of data parallelism (a set of clusters) combined with the advantages of a time-optimized framework based on the OST.

1.2. Motivating examples

In this section, we provide three examples that show the motivation of our work. In these examples, we reveal the need for having an intelligent mechanism relying between the user/application and a progressive analytics service.

Example 1. Imagine a Cloud computing setting where a service provider applies an intelligent monitoring service over the underlying resources. The provider wants to process numerous log data and performance metrics in order to have an insight on the overall performance of the system. This is very critical as consumers' requirements should be always satisfied because any malfunction could cause *Service Level Agreement* (SLA) violations. Any violation has economic consequences and, additionally, affects the reputation of the provider. Streams of values for the discussed metrics and logs are continually stored into the system. The provider, apart from simple alerting mechanisms, wants an analytics service over the huge amount of data in order to have: (a) a more complex and intelligent alerting mechanism and (b) a basis for scheduling future advances in the system. The first issue requires the execution of queries over the stored data in real-time while the second is more 'relaxed' concerning the reception of each query outcome.

Example 2. A governmental agency provides access to huge amount of data related to various types (e.g., research data, sensor data). Such data are available to interested parties either for research or for commercial purposes. For each type of data, specific partitions are provided in order to facilitate data management. The agency also provides a reference point for these data, actually, a query executor serving a large number of users/applications.

Example 3. A bank, serving a large number of clients, manages their every day transactions. The bank needs a system for fraud detection that will respond and derive alerts in real-time. The huge amount of transactions are stored to a number of clusters that are continually updated. The fraud detection system should define queries over this large amount of data and wait for the response. Fraud should be detected in the minimum time consuming the minimum required resources.

In the above discussed scenarios, two issues are of great importance: (a) the separation of the huge amount of data into a number of pieces and, (b) an executor responsible to receive queries of a stream, process them and return the final results to users/applications. The data could be separated into a number of clusters, probably in different locations, and different processors undertake the responsibility of returning progressive results, for each cluster, in order to meet time constraints. Streams of data are connected to the underlying storage mechanisms and, thus, data are continually updated. Each query is executed over the entire set of clusters and when the 'appropriate' result is available, the final outcome is delivered to the upper level application. The executor is responsible to return a response in the minimum amount of time. Accordingly, the executor asks for the execution of the query from the underlying set of processors. The selected set of processors could not be exhaustive, as the executor decides in real-time the plan for assigning the query execution to specific processors and for retrieving the final responses. The executor does know how the data are separated and, thus, it asks for progressive results accompanied with the confidence interval that every processor provides on these early (partial) results. The executor, by serving a large number of requests, should decide the right time to aggregate partial results and return the final outcome to the user/application. Hence, it saves time and resources increasing its throughput. Through this approach, the executor releases computational resources and can serve much more users/applications. It should be noted that the final outcome is returned to the user/application only when the executor is sure about the quality of the final outcome. The quality of the final outcome is affected by the confidence values for every set of early (partial) results.

1.3. Related work

Data are continuously collected in many application domains like financial services, life sciences, mobile services, etc. On-line users create huge amount of content like blog posts, tweets, social networking interactions or photos [32]. Big data analytics involves the process of collecting, organizing and analyzing big data. The aim is to discover patterns, especially in the case of unstructured data. Hence, meaningful information could be extracted consisting of the basis of more complex decision making mechanisms. Successful decision making mechanisms will increasingly be driven by analytics-generated insights.

A number of tools for big data analytics have been proposed in the literature. The majority of them concern batch oriented systems and they build on top of the Hadoop. A number of research efforts try to reveal performance insights to the discussed framework [2,12,23]. Researchers try to provide new functionality on top of the Hadoop in order to enhance the performance of the proposed systems. For instance, the authors in [19] propose Starfish which is a self-tuning tool for big data analytics. It can be adapted

Please cite this article in press as: K. Kolomvatsos et al., An Efficient Time Optimized Scheme for Progressive Analytics in Big Data, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.02.001

to user needs and to the system workload in order to provide an efficient solution. The proposed architecture adopts a self-tuning database system in order to be adapted to changes in users' requirements.

When big data applications involve continuous queries, for having near real-time responses, such applications include a large number of clusters or machines. However, retrieving a response near real-time could be very difficult due to limitations defined by the number of data and the underlying hardware performance. Querying data samples and the provision of progressive analytics is an efficient solution to the described problem [3]. In addition, the automatic selection of data samples is a very difficult task. Samples should be defined taking into consideration the domain, the underlying infrastructure and so on. Specific sampling techniques have been proposed in the literature [8,18,27]. In progressive analytics the *Approximate Query Processing* (AQP) technique is the key mean for handling the accuracy in early (partial) results. AQP aims at providing confidence intervals for early results and select processing orders to bias [8,13,29]. Users defining queries are not involved in the process, however, based on the confidence intervals could develop an intelligent mechanism for handling this information. When accuracy is at acceptable levels (according the specific application domain), the process could be stopped.

A progressive analytics system is presented in [5]. The system is based on a framework called Prism and allows users to communicate progressive samples to the system. Queries are processed over the defined samples. The authors proposed Now! a progressive data-parallel computation framework for Windows Azure, where progress is understood as a first-class citizen in the framework. Now! mainly works with streaming engines to support progressive SQL over big data. In [9], the authors present an on-line MapReduce scheme that supports on-line aggregation and continuous queries. For decreasing the latency of the system, the authors propose to have the Map task sending early results to the Reduce tasks. This mechanism enables the generation of approximate results, which is particularly useful for interactive analytics scenarios. In [24], the authors present a continuous MapReduce model. The execution of the Map and Reduce functions is coordinated by a data stream processing platform. Latency is improved through a model where mappers are continually fed by data instead of files and the retrieved results are transferred to reducers.

CONTROL [18] is an AQP system adopted to support progressive analytics. Users have the opportunity to refine answers and have on-line control of processing. This way, users are actively involved in the data analysis process. DBO [22] is another AQP system able to compute the exact answer to queries over a large relational database in a scalable fashion. DBO can have an insight on the final response together with specific bounds for the accuracy of the early results. As more information is processed, the DBO has the opportunity to provide more accurate results. Users can stop the process at any time, if the accuracy level is of their preference.

1.4. Contribution and organization

Our aim is to provide an intelligent mechanism on top of a progressive analytics service. This mechanism will minimize users intervention in handling early (partial) results. In the literature, the majority of the proposed models involve users to manually define either data samples, where queries will be executed, or to stop the process when satisfied by the accuracy level of early results. However, this is very difficult, especially in very dynamic environments where big data streams is the common case. Let us consider a *Query Controller* (*QC*) that receives a stream of queries and performs their execution. Each query is assigned to a set of processors. The *QC* should provide the final outcome as soon as possible as it aims to serve a large number of users/applications. The sooner the final outcome is generated, the better the performance of the *QC* (throughput) and, respectively, of the applications. However, the process ends when the proposed system 'sees' that the quality of the final result is at high levels. The *Quality of Result* (*QoR*) depends on the confidence intervals that each processor returns to the *QC*. The higher the confidence is, the better for the system. We adopt the AQP technique and a progressive analytics approach. The provision of confidence intervals for partial results is responsibility of the AQP system [5,7,14] and not of the proposed mechanism. The proposed mechanism could be applied either for batch oriented models or for stream processing models.

Table 1
Concept notations.

Notation	Description
QC	Query controller
QoR	Quality of result
$\mathcal{Q}$	Query under consideration
$\mathbb{Q}_E$	Partial query results
P_i	Query processors ($i = 1, 2, \ldots$)
CI_i	Confidence Interval for P_i
C_i	Confidence value for CI_i
c_i	Confidence value realization for CI_i
t^*	Optimal stopping time
Y_t	Reward at stage t
M_t	Maximum confidence value at stage t
Y^*	Reward at the optimal stopping time t^*
R^*	Optimal stopping rule
T^*	Principle of optimality
β	Discount factor for reward
$\mathbb{T}$	Discrete time domain

The paper is organized as follows. Section 2 presents our scenario and gives an insight to our setting. In Section 3, we present the proposed mechanism by giving details of our models. We describe analytical solutions and depict their realization. Performance metrics, simulation set-up and experimental evaluation are presented in Section 4. Finally, in Section 5, we conclude our paper by giving future extensions of our work.

2. Rationale and preliminary

In this section, we discuss the proposed architecture and present basic information about our setting. In Table 1, we give basic notation for our problem adopted throughout the paper accompanied by a short description.

2.1. The proposed architecture

Without loss of generality, we focus on the execution of a specific query $\mathcal{Q}$. The *QC* is responsible to manage the 'efficient' execution of $\mathcal{Q}$. With the term 'efficient', we depict the process that will result the best possible *QoR* in the minimum time. As mentioned, data parallelism can offer advantages in the 'efficient' query execution by splitting the data in to a number of pieces. We consider that no specific algorithm is adopted for splitting the data and, thus, we cannot be aware on the contents of each piece. Let us denote each piece of data as 'cluster'. As depicted in Fig. 1, for every cluster, a processor (P_i, $i = 1, 2, \ldots$) is responsible to manage the underlying data and return results to the *QC*.

Processors P_i adopt the discussed AQP technique and for every response, they also return the confidence interval CI_i [5] on the early (partial) results. The process, through which every P_i derives early results and the CI_i, is responsibility of the internal system and does not affect our model. We assume that P_i are assigned to execute the same query over different clusters adopting, probably, different execution methods. CI_i, actually, is an error estimate. Deriving a closed form for the CI_i is often a manual analytical process and, in the literature, is related to simple SQL queries [3]. Usually,

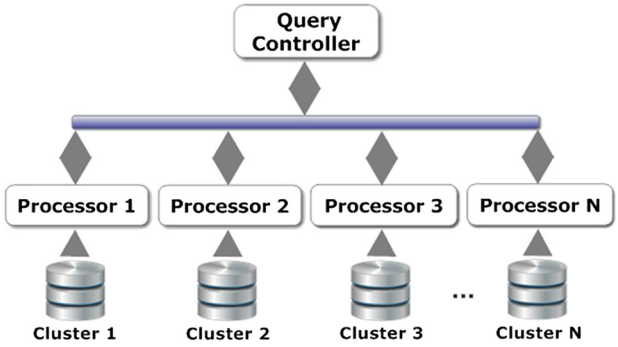


Fig. 1. The architecture of our model.

CI_i comes with the form of error bars defining the upper and the lower level of the error. Without loss of generality, we consider the variable C_i which depicts the estimate of the centre of the population described by the CI_i (confidence value). The calculation process of the C_i is beyond the scope of this paper.

Let us now consider the discrete time domain $\mathbb{T}$. In each $t \in \mathbb{T}$, the *QC* receives:

- the id of the processor P_i;
- early (partial) results for $\mathcal{Q}$ depicted by the set $\mathbb{Q}_E$;
- the confidence value C_i for these results.

As data are not separated by adopting an 'intelligent' method and our main scenario involves streams of data, C_i values could increase or decrease over time. This is because, confidence intervals could be shortened or expanded as more data are received and stored to the system. From the *QC* point of view, C_i values are received in a sequential order together with early (partial) results. The *QC* should wait to receive more results expecting an increased C_i, however, it should not wait for a long time as it should return the results to the user/application as soon as possible. Our aim is to provide an *optimal stopping model* that will result the *optimal time* where the *QC* should stop observing C_i values and return the aggregation of the partial results to the user/application.

2.2. Optimal Stopping Theory

The *Optimal Stopping Theory* (OST) [28], concerns finding the best time to take an action (decision) based on sequentially observed random variables. The final aim is to maximize an expected reward. The optimal stopping problem is defined by a sequence of random variables $C_1, C_2, \ldots$ whose joint distribution is known and a sequence of real-valued reward functions $Y_0, Y(x_1), Y(x_1, x_2), \ldots$. Let (Ω, B, P) be the probability space, and $\mathcal{G}_t$ be the sub-σ-field of B generated by $C_1, \ldots, C_t$. We have a sequence of σ-fields as $\mathcal{G}_1 \subset \mathcal{G}_2 \subset \ldots \mathcal{G}_t \subset B$. A stopping time is defined as a random variable $T \in 0, 1, \ldots, \infty$ such that the event $T = t$ is in $\mathcal{G}_t$. The aim is to choose an optimal stopping time t^* to maximize the expected future reward $E[Y_{t^*}]$. If there is no bound on the number of steps at which one has to stop, this is an *infinite horizon problem* and the optimal return can be calculated via the optimality equation. When there is a known upper bound on the number of steps, it is a *finite horizon problem* and the optimal return can be solved by backward induction.

3. The proposed time-optimized mechanism

3.1. Model description & problem formulation

Every P_i sends early (partial) results and C_i to the *QC*. We consider the random independent variables $C_i, i = 1, 2, \ldots$ with realizations $c_i, i = 1, 2, \ldots$. The *QC* observes $c_i \in [0, 1]$ and decides the **optimal stopping time** t^* where the *QC* stops the process and deliver the aggregated results to the user/application. At every stage $t \in \mathbb{T}$, the *QC* receives c_t and checks if the current reward is greater than the expected future reward. The reward for stopping at t is

$$Y_t = \beta^t M_t \tag{1}$$

where

$$M_t = max\left(c_t^*\right) \tag{2}$$

and $c_t^* = \{c_1, c_2, \ldots, c_t\}$. $\beta \in (0, 1)$ affects the *QC* behaviour as follows. The *QC* probably should delay the decision in anticipation of a better c_i when $\beta \to 1$. It should not delay the decision when the user/application (e.g., a critical application) requires an immediate response to $\mathcal{Q}$ ($\beta \to 0$). If the *QC* never stops, the reward is considered equal to zero, thus, we assume $Y_0 = Y_\infty = 0$. The discount factor β defines an upper limit on the stages for delivering the final outcome.

Problem 1. Identify the optimal stopping time t^* where the expected *QC* reward is maximized.

Problem 2. Find an optimal stopping rule, such that the *QC* terminates the process in order to maximize the expected reward Y_t, i.e., $E[Y_t]$.

We can treat Problem 1 as a finite or as an infinite horizon problem. In the finite horizon case, the *QC* should respond in a specific time interval while in the infinite horizon case the *QC* receives c_i and it has no 'pressure' on the time for the final decision. However, the discount implied by the parameter β makes the *QC* to return the final result in a rationale time interval. The reason is that β affects the reward that the *QC* tastes at future stages. It should be noted that we consider a model involving 'recall', meaning that the *QC* stores the responses and early results up to stage t.

Once the *QC* receives c_t, it decides whether to continue the process or not, by examining the expectation of the future reward, i.e., based on the $\max\{m, c_t\}$ value, with $m = \max\{c_0, c_1, \ldots, c_{t-1}\}$. In other words, the *QC* has to find an optimal stopping time (stage t^*) at which the supremum in Eq. (3) is attained.

$$\sup_t E[Y_t] \tag{3}$$

Suppose that at some stage t, the *QC* has received $M_t = m'$ and it is optimal to continue the process. Then, at the next stage $t + 1$, if M_{t+1} is still m', because $c_{t+1} \leq m'$, it is optimal to continue due to the invariance of the problem in time [15]. Hence, based on the principle of optimality, this problem can be solved as an optimal stopping problem with discounted future reward and without recall. This means that the reward, in Eq. (2), can be considered as $Y'_t = \beta^t M_t$ and the problem in Eq. (3) assumes the same solution as the following problem:

Problem 3. Find a t^* such that the $\sup_t E[Y'_t]$ is attained.

3.2. Model analysis

Let rewards $Y_0, Y_1, \ldots, Y_\infty$ where $Y_t = f(c_1, c_2, \ldots, c_t)$. The sequence $\langle c_t, \mathcal{F}_t \rangle$ is defined by a probability space Ω, an increasing sequence of sub σ-algebras $\{\mathcal{F}_t\}_1^\infty$, the sequence of random variables C_i ($c_i \in \mathcal{F}_t$) and $E[c_i]$, $\forall i$. The following two assumptions should be true in order to have an optimal stopping time:

(A) $E[sup_t Y_t] < \infty$
(B) $lim\, sup_{t\to\infty} Y_t \leq Y_\infty$

Definition 1. A stopping rule R^* is the rule for which the following inequality stands true: $E[Y_{R^*}]|\mathbb{F}_t) > Y_t$ a.s. on $\{R^* > t\}$, $\forall t$.

Theorem 3.1. *Assuming condition (A), for any stopping rule R^*, there is a regular stopping rule R' such that $E[Y_{R'}] \geq E[Y_{R^*}]$.*

Proof. See [15]. □

Theorem 3.2. *Under (A) and (B) conditions, there exists a stopping rule R^* such that $E[Y_{R^*}] = M^*$ where $M^* = sup_R E[Y_R]$.*

Proof. See [15]. □

Theorem 3.3. *Under (A), if an optimal stopping rule exists, in particular, if (B) holds true, then R^* is optimal.*

Proof. See [15]. □

Theorem 3.4. *For the model defined by Eq. (2) and Eq. (3), an optimal stopping time exists.*

Proof. We have $Y_t \leq \beta^t max(c_1, c_2, \ldots, c_t) \leq max(\beta c_1, \beta^2 c_2, \ldots, \beta^t c_t) \leq \sum_{j=1}^{\infty} \beta^j |c_j|$. Additionally, we have $E[sup_t Y_t] \leq \sum_{j=1}^{\infty} \beta^j \times E[|c_j|] = \frac{\beta}{1-\beta} E[|c_j|] < \infty$. Based on the above, condition (A) is satisfied.

Additionally, $lim\, sup_t Y_t \leq lim_t \beta^t \sum_{j=1}^{t} |c_j| = lim_t \beta^t \frac{\sum_{j=1}^{t} |c_j|}{t}$. From the law of large numbers, we take $\sum_{j=1}^{t} |c_j| \to E[|c|]$ and $t\beta^t = 0$. Hence, $lim\, sup_t Y_t \leq Y_\infty = 0$ and condition (B) is satisfied. □

Based on the above, the optimal stopping rule and the optimal stopping time is given by the principle of optimality i.e.,

$$T^* = \min\left\{t \geq 0 : Y_t \geq Y^*\right\} \tag{4}$$

3.2.1. Finite horizon model

In the finite horizon case, there is an upper limit of stages equal to N. Till stage N, the *QC* should return the final outcome to the user/application. As the optimal stopping rule exists as well as the optimal stopping time t^*, we can focus on the optimal stopping rule and solve our problem through backward induction. Let us define $J_t(c_t) = max\left(\beta^t c_t, E[J_{t+1}(c_{t+1})]\right)$. We take $Z_t = \frac{J_t(c_t)}{\beta^t}$ and, thus,

$$Z_t(c_t) = max\left(c_t, \beta E[Z_{t+1}(c_{t+1})]\right) \tag{5}$$

If we take $Z_t(x) = max(x, a_{t+1})$, we, finally, have that $a_t = \beta E[Z_t(x)] = \beta E[max(x, a_{t+1})]$. Remind, that every $c_i \in [0, 1]$ and, thus, the optimal stopping time is defined at the time where the current reward Y_t is greater than a_t where

$$a_t = \beta \left(\int_0^{a_{t+1}} a_{t+1} dF(c) + \int_{a_{t+1}}^{1} c dF(c) \right) \tag{6}$$

with

$$a_N = \beta \mathbb{E}(c) \tag{7}$$

Solving Eq. (6), we can have the recursive equation that gives us the reward limit, for every stage t. Above this reward limit, the *QC* stops observing c_i and returns the final outcome to the user/application.

3.2.2. Infinite horizon model

In the infinite horizon case, the optimal stopping time t^* is given by Eq. (4) where

$$Y^* = E[max\left(\beta c_t, Y^*\right)] \tag{8}$$

Following [15], we have that $Y^* = E[max(\beta c_1, Y^*)]$ and, thus, $Y^* = \beta E[max(c_1, Y^*)]$ Finally, as $c_i \in [0, 1]$, we have that Y^* is the solution of the following equation:

$$Y^* = \beta \left(\int_0^{Y^*} Y^* dF(c) + \int_{Y^*}^{1} c dF(c) \right) \tag{9}$$

3.3. Model realization

Our model realization depends on the selection of the probability distribution for variables C_i. In this section, we choose two probability distributions and solve our model. The aim is to reveal the benefits of each one in the discussed setting. We adopt: (a) the *Uniform* distribution and (b) the *Exponential* distribution. Frequentist inference involves inference mechanisms based on confidence distributions. As defined in [31], the confidence distribution for a parameter θ involves the case where at θ, the distribution follows Uniform and, thus, it is not informative in direction [31]. Example application domains, where Uniform is adopted, include Econometrics [33] or participatory sensing [30]. In [20,21], the authors assume that confidence values follow an Exponential distribution. These efforts propose models for combining classifiers for character recognition.

When applying the Uniform distribution, we assume that c_i values are of equal probability in a specific interval. In other words, c_i values have the same probability to be observed by the *QC*. On the other hand, by applying the Exponential distribution, we aim to handle various cases where the *QC* assumes that confidence values will be low or high affected by the rate of the Exponential. As no special data separation mechanism is adopted, the *QC* cannot be sure about the level of the confidence values. It should be noted that there is no reason to adopt a distribution 'favourite' to large values ($c_i \to 1$) as in these cases, the intelligent mechanism is useless. Large confidence values depict the case where each processor believes that partial results are similar to the final, from the beginning of the $\mathcal{Q}$ execution. However, this scenario cannot be representative when we consider streams of data feeding the system.

3.3.1. Finite horizon model

Proposition 3.5. *If confidence values follow a Uniform distribution, the recursive equation $a_t = \frac{\beta}{2}\left(a_{t+1}^2 + 1\right)$ defines values that indicate the optimal stopping rule at every stage t.*

Proof. Applying the probability density function (PDF) of the Uniform in Eq. (6), we can easily take that $a_t = \frac{\beta}{2}\left(a_{t+1}^2 + 1\right)$. Additionally, through Eq. (7), we get $a_N = \frac{\beta}{2}$. □

Proposition 3.6. *If confidence values follow an Exponential distribution, with rate λ, the recursive equation $a_t = \frac{\beta}{\lambda}\left(\lambda a_{t+1} + e^{-\lambda a_{t+1}} - (\lambda + 1)\, e^{-\lambda}\right)$ defines values that indicate the optimal stopping rule at every stage t.*

Proof. Applying the PDF of the Exponential distribution in Eq. (6), we can easily take that $a_t = \frac{\beta}{\lambda}\left(\lambda a_{t+1} + e^{-\lambda a_{t+1}} - (\lambda + 1)\, e^{-\lambda}\right)$. Additionally, through Eq. (7), we get $a_N = \frac{\beta}{\lambda}$. □

3.3.2. Infinite horizon model

Proposition 3.7. *If confidence values follow a Uniform distribution, the QC should stop the process and return the aggregation of the partial results when the reward at stage t is greater than* $Y^* = \frac{1-\sqrt{1-\beta^2}}{\beta}$.

Proof. Solving the integrals in Eq. (9) and, accordingly, the equation $\frac{\beta}{2}Y^{*2} - Y^* + \frac{\beta}{2} = 0$ for Y^*, we get $Y^* = \frac{1-\sqrt{1-\beta^2}}{\beta}$. □

Proposition 3.8. *If confidence values follow an Exponential distribution, with rate* λ*, the QC should stop the process and return the aggregation of the partial results when the reward at stage t is greater than* $Y^* = \frac{\omega+(\beta-1)LambertW(0,-\frac{\beta e^{-\frac{\omega}{\beta-1}}}{\beta-1})}{\lambda(\beta-1)}$ *where* $\omega = \beta e^{-\lambda}(\lambda+1)$.

Proof. Solving the integrals in Eq. (9) and, accordingly, the equation $Y^* - \frac{\beta}{\lambda}\left(\lambda Y^* + e^{-\lambda Y^*} - (\lambda+1)e^{-\lambda}\right) = 0$ for Y^*, we get $Y^* = \frac{\omega+(\beta-1)LW(0,-\frac{\beta e^{-\frac{\omega}{\beta-1}}}{\beta-1})}{\lambda(\beta-1)}$, where $\omega = \beta e^{-\lambda}(\lambda+1)$. *LW* indicates the Lambert function (also known as the omega function or product logarithm). Hence, *LW*() represents the solution of the following equation:

$$W(x)e^{W}(x) = x \tag{10}$$

with $x = -\frac{\beta e^{-\frac{\omega}{\beta-1}}}{\beta-1}$. □

4. Experimental evaluation

We elaborate on the performance of the optimal stopping models. Through a large number of experiments, we evaluate the **finite horizon model** OSM_F as well as the **infinite horizon model** OSM_I. Various simulation scenarios are adopted in order to evaluate the performance of the OSM_F and the OSM_I when, in our models, we adopt the Uniform or the Exponential distribution for depicting the confidence values. When the Uniform is adopted, we aim to handle scenarios where the *QC* receives confidence values in an 'agnostic' manner. The term 'agnostic' means that the *QC* considers that confidence values, in the sample space, have an equal opportunity of occurring. The *QC* does not know anything about the process followed by every processor P_i and, additionally, it believes that confidence values could continually be changed either in an increasing or in a decreasing manner. When we adopt the Exponential distribution, we focus on high or low confidence values based the λ parameter. For instance, when $\lambda = 1.0$, the *QC* considers that it will receive high confidence values. The higher the λ is, the smaller the confidence values become. In this setting, we aim to evaluate different scenarios where streams of data affect the confidence values and, thus, the query process at every cluster of data. When $\lambda \to 1.0$, we assume that data received by the stream have small 'fluctuations' and each processor could easily conclude high confidence values on the partial results. When λ is very high, we assume that data have 'heavy fluctuations' and processors cannot easily conclude a specific high confidence value.

4.1. Performance metrics & simulation set-up

We report on the performance of the proposed models concerning the throughput and the maximum *QoR*. In this paper, we consider the throughput as the amount of the successful queries executed in a specific amount of time. We define metric $\mathcal{T}$, that depicts the throughput of the *QC*. $\mathcal{T}$ is defined by the following equation:

$$\mathcal{T} = \frac{|\mathcal{Q}|}{\sum_{k=1}^{|\mathcal{Q}|} T_E} \tag{11}$$

where $\mathcal{Q} = \{Q_1, Q_2, \ldots\}$ are the queries executed by the *QC* and T_E is the total execution time for each Q_i. $\mathcal{T}$ represents the number of queries executed by the *QC* divided by the total time required for the execution of those queries. Actually, $\mathcal{T}$ depicts the number of queries executed over a fixed time (milliseconds in our case). The higher the $\mathcal{T}$ is, the higher the performance of the proposed models becomes. $\mathcal{T}$ is a very important metric as in our setting, we consider continuous queries execution and aim to have high throughput performance.

The *QoR* for each query is evaluated by the γ metric which is defined by the following equation:

$$\gamma = \frac{\sum_{k=1}^{|\mathcal{Q}|} C^*}{|\mathcal{Q}|} \tag{12}$$

where $C^* = max(c_1, c_2, \ldots, c_{t^*})$. The γ metric depicts the maximum confidence value received till the stopping time t^*. The higher the γ is, the higher *QoR* the *QC* tastes (and the user/application, respectively). High c_i indicates that processors P_i are confident on the partial results retrieved by the progressive analytics process.

For comparison purposes, we define $\mathcal{T}_D$ and γ_D metrics. The following equations hold true:

$$\mathcal{T}_D = \frac{\mathcal{T}_F - \mathcal{T}_I}{\mathcal{T}_I} \cdot 100\% \tag{13}$$

$$\gamma_D = \frac{\gamma_F - \gamma_I}{\gamma_I} \cdot 100\% \tag{14}$$

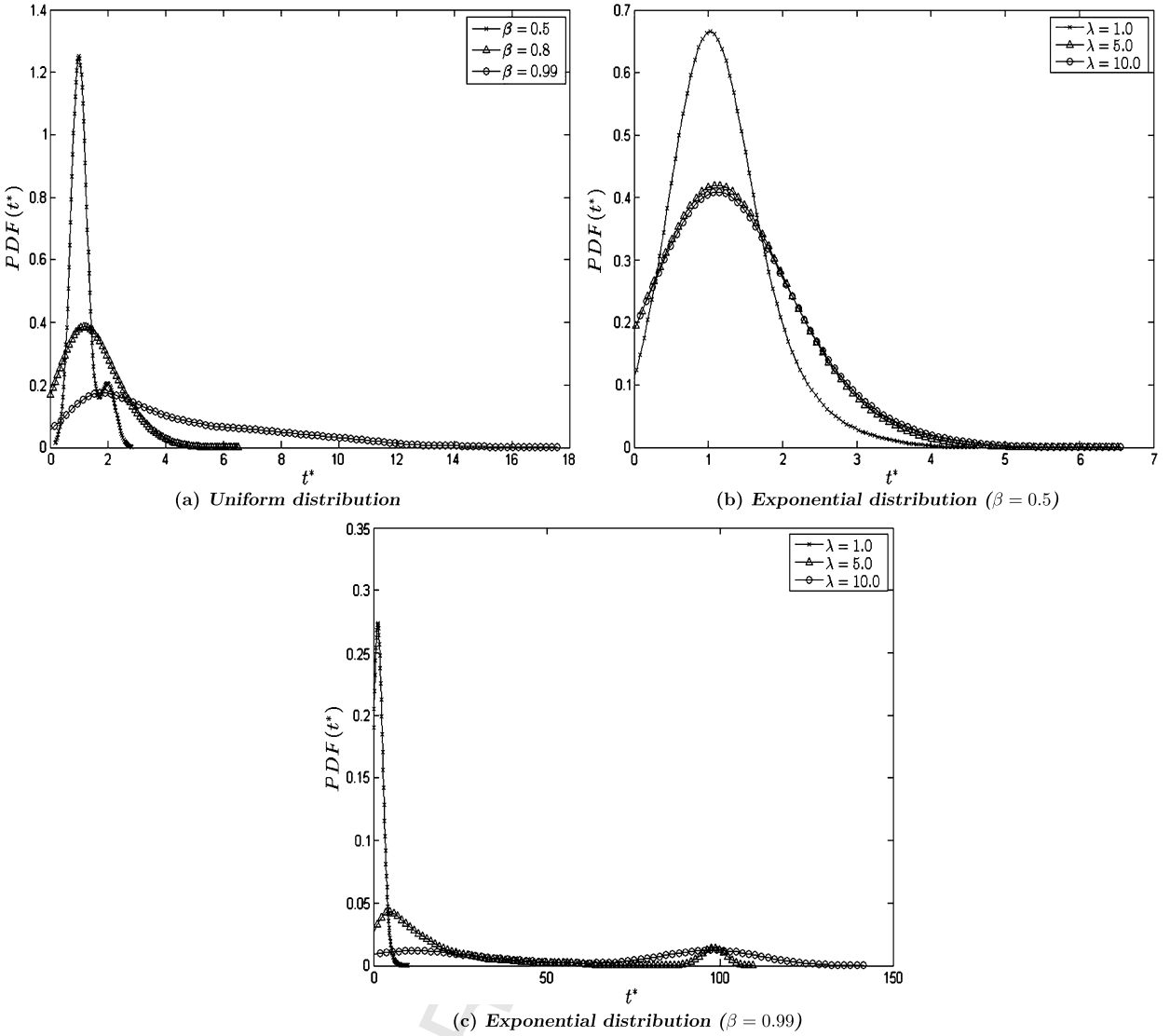
where $\mathcal{T}_F$ is the throughput of the OSM_F and $\mathcal{T}_I$ is the throughput of the OSM_I. Additionally, γ_F is the maximum confidence result of the OSM_F and γ_I is the maximum confidence of the OSM_I. These metrics are adopted to depict the difference in the performance between the finite and the infinite horizon models. Our aim is to reveal the advantages and weaknesses of the proposed models for the same simulation scenarios.

We compare the proposed optimal stopping models with a deterministic model. Let $t^{\#}$ be the stopping time where the reward exceeds a pre-defined threshold h for the **Deterministic Stopping Model** (*DSM*). We compare OSM_F and OSM_I with the *DSM* in order to demonstrate the optimality achieved by the proposed models. Specifically, a mechanism that is based on a *DSM* proceeds with a stopping decision (i.e., decision **D2**) iff the discounted reward at time t, $Y_t = y$, exceeds a fixed threshold h. The *DSM* decides as follows:

- **D1** Continue receiving partial results and confidence values at the next time slot $t+1$, if $Y_t < h$;
- **D2** Stop and proceed with aggregating partial results and return them to the user/application, if $Y_t \geq h$.

We evaluate our models with the *DSM* for diverse values of h, in order to examine the cases where OSM_F and OSM_I results in better rewards. The evaluation of each mechanism (OSM_F, OSM_I and *DSM*) refers to the average throughput and maximum confidence values for a large number of experiments.

Without loss of generality, we consider the sampling period $d = 1$ time unit. For each experiment, we run 1000 runs and take results for the OSM_F, OSM_I and *DSM*. We take $\beta \in \{0.2, 0.4, 0.6, 0.8, 0.99\}$ and $\lambda \in \{1.0, 5.0, 10.0, 20.0, 30.0\}$. $\beta \to 0.0$ represents the scenario where the *QC* should immediately return the final result to the user/application while $\beta \to 1$ represents the scenario where the *QC* has not 'anxiety', defined by time restrictions, on returning the final outcome. In the second case, the *QC*



(a) *Uniform distribution* (b) *Exponential distribution* ($\beta = 0.5$)

(c) *Exponential distribution* ($\beta = 0.99$)

Fig. 2. Pdf of t^* for the OSM_F model.

could wait for longer to receive more confident partial results. Similarly, as described, low λ values are adopted when we deal with the scenario where the *QC* receives large confidence values and the opposite stands for high λ. Finally, we take $h \in \{0.5, 0.9\}$. When $h = 0.5$, the *DSM* stops the observation process for relatively low confidence values compared to the scenario where $h = 0.9$.

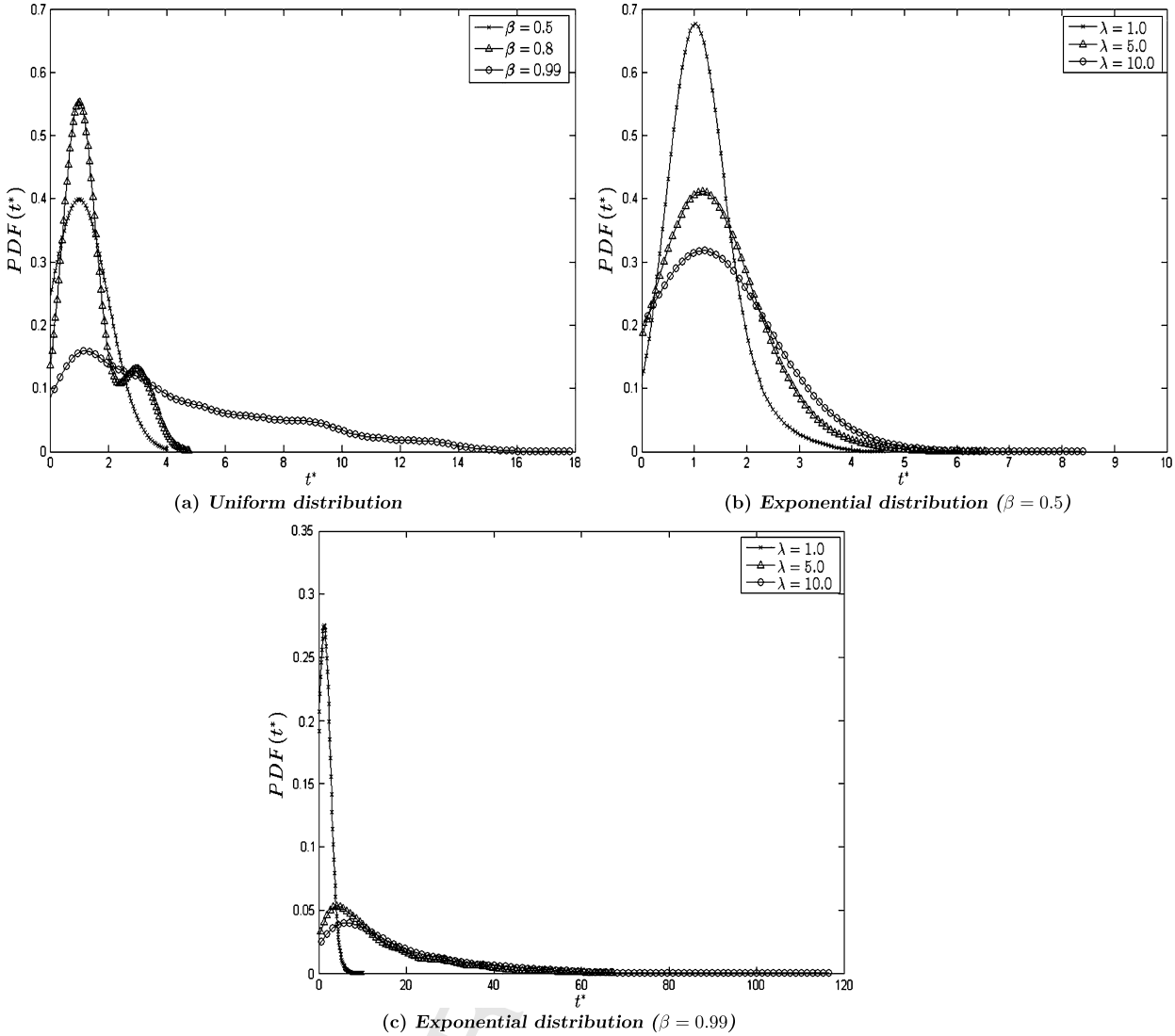
4.2. Performance assessment

We initially report on an estimate of the PDF for the proposed models. In Fig. 2, we present an estimate of the PDF of t^* for the OSM_F model. We observe that the stopping time is affected by the parameter β. The higher the β is, the higher the t^* (in average) becomes. The reason is that the proposed mechanism waits to receive higher confidence values before it decides to stop the process, aggregate partial results and return the final outcome to the user/application. In such cases, the reward at $t + 1$ stage remains at similar levels as at t. When $\beta \to 0$, the *QC* reward is highly reduced as the decision for stopping the observation process is delayed. In general, the adoption of an Exponential distribution for depicting confidence values results higher t^* especially when $\beta = 0.99$ and $\lambda = 10.0$. $\lambda = 10.0$ leads to very low confidence values and, thus, the *QC* should delay the decision of stopping the process as it expects higher confidence in the upcoming stages. When $\beta = 0.5$, the mechanism results a very low t^* as the reward at time t is half of the reward at time $t - 1$. When Exponential is adopted, the proposed model is more sensitive to β and, thus, the final aggregation is immediately decided after receiving the first partial results (e.g., t^* is very low, below 7, when $\beta = 0.5$).

In Fig. 3, we see our results for the OSM_I model. We observe similar results as in the OSM_F with slightly lower t^*. Recall that in the OSM_I, the mechanism does take the final decision having in mind a specific horizon to conclude the aggregation result. It should be noted that results depicted by Fig. 2 and Fig. 3 represent the probability density estimate over the observed values retrieved by our simulations. Moreover, our models do not result a 'stopping' action when $t^* = 0$. In this case, the *QC* should stop before it receives the first partial results. We assume that the stopping decision is taken at $t^* \geq 1$. A future extension of our model is to involve a warm up period for the discussed setting after which the *QC* could stop the process and return the final outcome to the user/application. However, an intelligent mechanism for deriving the appropriate warm up period is necessary in this setting.

We compare the performance of the OSM_F and the OSM_I concerning the throughput of the proposed mechanisms and the *QoR* of the final outcome. In Fig. 4, we see our results when Uniform distribution is adopted. The OSM_I exhibits better throughput, how-

Please cite this article in press as: K. Kolomvatsos et al., An Efficient Time Optimized Scheme for Progressive Analytics in Big Data, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.02.001



(a) *Uniform distribution* (b) *Exponential distribution* ($\beta = 0.5$)

(c) *Exponential distribution* ($\beta = 0.99$)

Fig. 3. Pdf of t^* for the OSM_I model.

ever, it performs worse than OSM_F concerning the γ metric. The reason is that the finite horizon model delays to take the stopping decision till the upper limit for stages N. As confidence values follow the Uniform distribution, the infinite model is heavily affected by β and, thus, when the model achieves a limited reward stops the process. The main reason is the 'pessimistic' approach that the infinite model follows. As the confidence values are considered of equal probability, the infinite horizon model, the first time that it observes a reward above Y^*, stops the process. The finite horizon model is forced to receive partial results till N, especially when $\beta \to 1$. When $\beta \to 0$, both models exhibit a similar behaviour concerning the throughput as the decision should be immediately taken due to the reason that the future reward is eliminated. As $\beta \to 1$, the finite model results decisions at stages close to the upper limit N in order to achieve better performance. There is a trade off between the two proposed models. If the *QC* wants to serve more queries, the OSM_I should be chosen. If the *QoR* is the main focus of the system, the OSM_F is the best solution. The higher the β is, the higher the difference in the performance becomes.

The opposite results are obtained when an Exponential distribution is adopted (Figs. 5 and 6). In this setting, the OSM_F performs better than the OSM_I either for the $\mathcal{T}_D$ or the γ_D metric. When $\beta \leq 0.5$, both OSM_F and OSM_I models exhibit the same performance concerning the γ metric. In general, the Exponential distribution has an 'attitude' to low values based on the λ parameter. In this case, the infinite model decides the stopping action at later stages compared to the finite horizon model that should take the decision before N. Hence, the throughput is lower for the infinite horizon model. On the other hand, a delay in the decision of the infinite horizon model may increase the *QoR* as the proposed model receives more confidence values (in number) and chooses the optimal among them. It should be noted that the discussed process is heavily affected by the 'nature' of the Exponential distribution. When $\lambda \to \infty$, the Exponential distribution leads to low values in general, and, thus, the infinite horizon model delays more the stopping decision decreasing the throughput. On the other hand the infinite horizon model for low λ and $\beta \to 1$ leads to an increased *QoR* as it selects the optimal *QoR* from a large set of values (compared to the finite horizon model). When $\lambda \to 0$, the Exponential distribution does not exhibit an 'attitude' to low values and, thus, the proposed models have similar performance, especially when $\beta \to 0$. The finite horizon mechanism seems to be the best solution when we deal with the Exponential distribution for depicting confidence values.

We compare the proposed models OSM_F and OSM_I with the deterministic model *DSM*. In our experiments, we adopt $h = 0.5$ and $h = 0.9$. In Fig. 7, we present our results for the $\mathcal{T}$ metric while in Fig. 8, we give results for the γ metric. The optimal stop-

Please cite this article in press as: K. Kolomvatsos et al., An Efficient Time Optimized Scheme for Progressive Analytics in Big Data, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.02.001

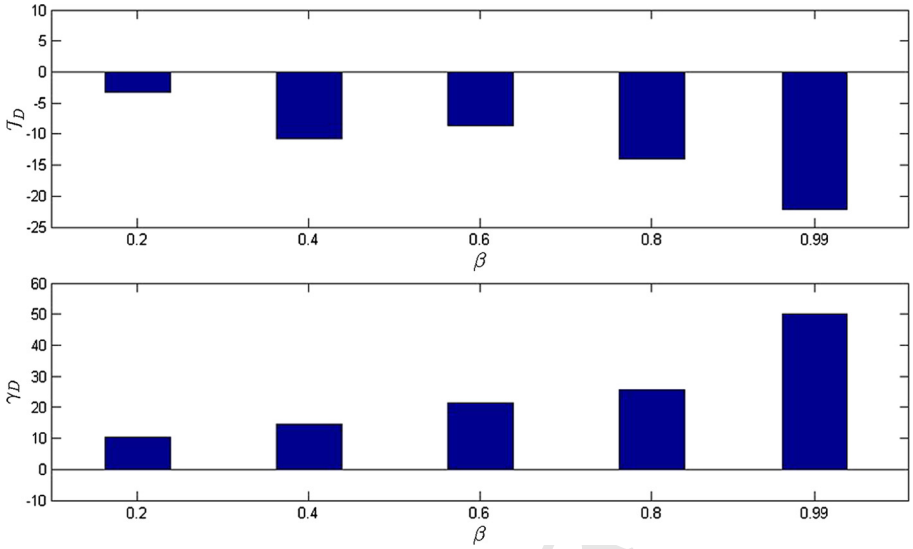


Fig. 4. OSM_F vs OSM_I (Uniform distribution).

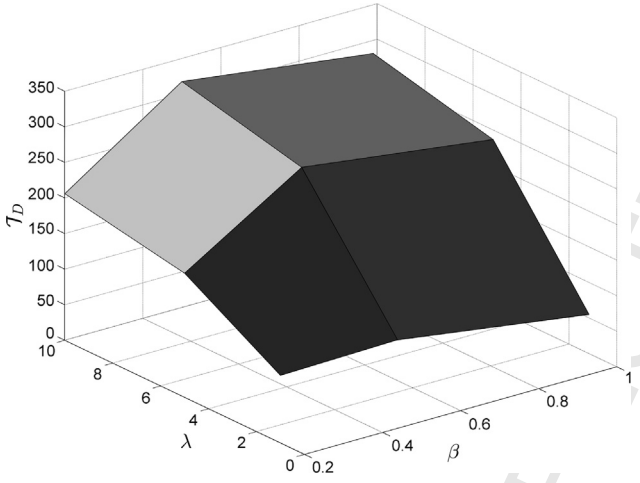


Fig. 5. OSM_F vs OSM_I (Metric: $\mathcal{T}$, Exponential distribution).

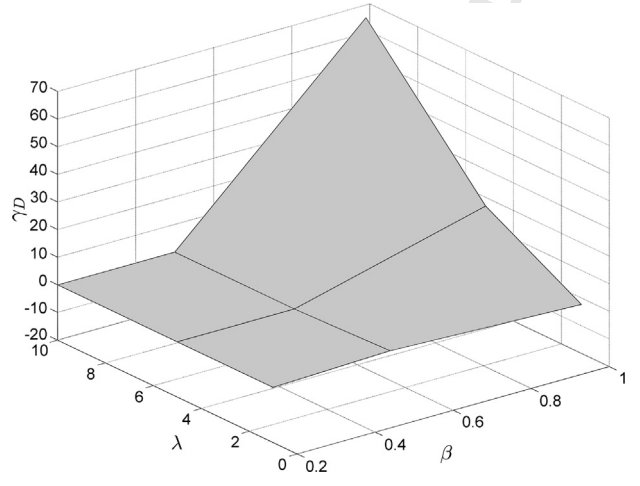


Fig. 6. OSM_F vs OSM_I (Metric: γ, Exponential distribution).

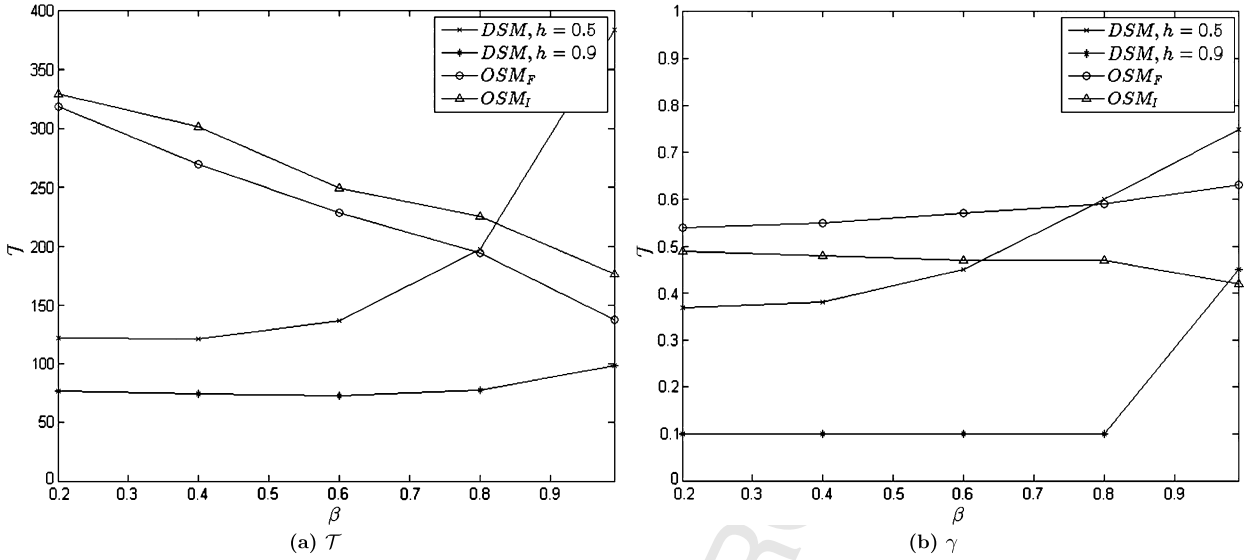
ping models perform better than the *DSM* in the majority of our experiments. The *DSM* has better throughput results only when $\beta \to 0.99$, however, for $h = 0.5$. A threshold $h = 0.5$, probably, is very low to be adopted in real scenarios. Moreover, recall that when $\beta = 0.99$, the proposed mechanism, mainly, retains the current reward (approximately) when receiving successive confidence values. Hence, no 'anxiety' is exercised in the proposed mechanism. The OSM_I exhibits better throughput results while the OSM_F gives better γ values. The *DSM* adopting a high h value (i.e., $h = 0.9$) exhibits the worse performance compared to the rest models. This is natural as the *DSM* waits to receive the confidence values that will result reward over this high threshold. The above described results are related to the adoption of a Uniform distribution for the confidence values.

In Fig. 8, we present results for a large number of experiments combining different β, λ and h values when the Exponential distribution is adopted to depict confidence values. As natural, the examined models are affected by λ. Concerning the $\mathcal{T}$ metric, when $\lambda \to 1.0$, the *DSM* exhibits higher throughput compared to the proposed *OSM* techniques. In this case, processors return high confidence values for their partial results and, thus, the process could be immediately stopped. When $\lambda > 5.0$, the OSM_F performs better than the rest models ($\mathcal{T}$ metric). The OSM_F and the OSM_I achieves higher γ values for $\beta = 0.99$ and $\lambda \geq 5.0$. In such cases, the proposed mechanism results large t^* as they wait to receive high confidence values before deciding to stop the observation process and return the final outcome to the user/application.

5. Conclusions and future work

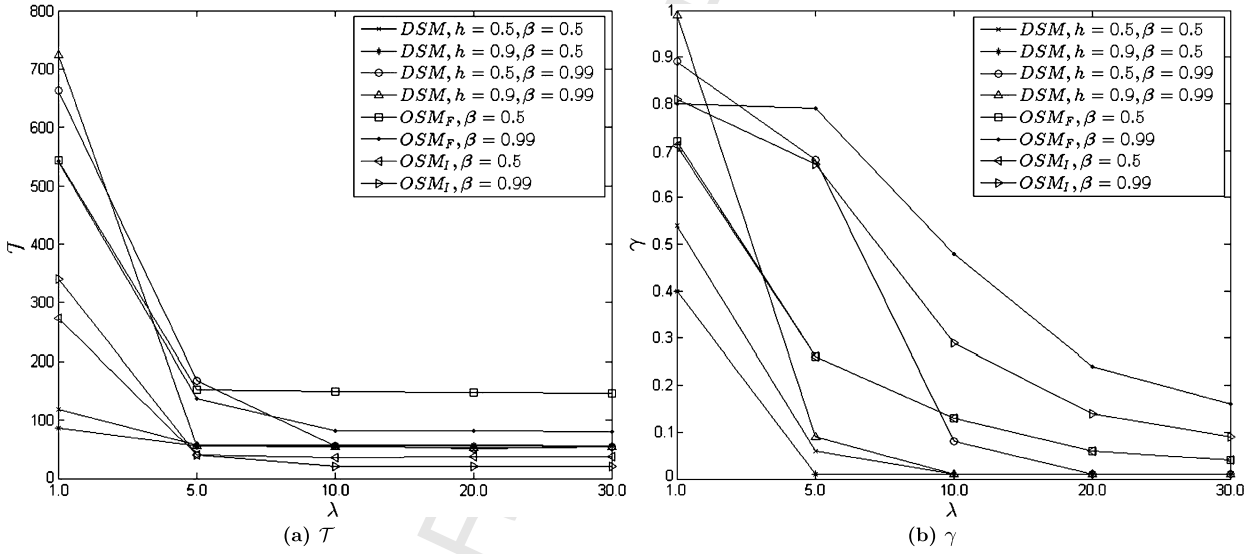
Progressive analytics can offer many advantages when adopted to manage big data. Such technique could be very efficient, especially when streams of data is the main scenario. In such cases, data are continually updated and, thus, there is not any insight on their form. In this paper, we focus on data parallelism and assume an underlying progressive analytics service. We propose a mechanism for handling responses retrieved by processors querying clusters of data. Each processor adopts a progressive analytics scheme and is responsible to return early (partial) results and a confidence value to our mechanism. We adopt the principles of the Optimal Stopping Theory (OST) and model the behaviour of a Query Controller (*QC*) responsible to manage multiple queries. We build on top of the processors and provide an intelligent decision making mechanism. Our aim is to alleviate users/applications from the responsibility of monitoring continuous results retrieved by processors and deciding when it is the right time to stop the process in order to save time and resources. Two models are described: the first assumes a finite horizon scheme while the second

Please cite this article in press as: K. Kolomvatsos et al., An Efficient Time Optimized Scheme for Progressive Analytics in Big Data, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.02.001



(a) $\mathcal{T}$ (b) γ

Fig. 7. OSM vs DSM (Uniform distribution).



(a) $\mathcal{T}$ (b) γ

Fig. 8. OSM vs DSM (Exponential distribution).

considers an infinite horizon setting. A large number of experiments reveal the efficiency of the proposed models. We focus on the throughput of the *QC* when working in a continuous query scenario and on the quality of the final outcome. Through our results, it is revealed that there is a trade off between throughput and the quality of the final outcome.

Future extensions of our work include the definition of an intelligent scheme for creating plans and resulting assignments of queries to specific processors. Every query will be assigned to specific processors, probably, a subset of the processors available to the *QC*. For this, we are going to provide specific models for queries and processors characteristics. Through this approach, the efficiency of the proposed system will be maximized as the appropriate processors will be selected only for those queries that their performance will be the maximum. A learning technique will be also adopted to build an intelligent scheme for assigning queries to processors. For this, modelling the underlying data and the adoption of an algorithm that splits them to the appropriate pieces, in the most efficient way, seem to be imperative.

Uncited references

[16]

References

[1] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S.B. Zdonik, Aurora: a new model and architecture for data stream management, VLDB J. 12 (2) (2003).
[2] A. Abouzeid, K. Bajda-Pawlikowski, D.J. Abadi, A. Rasin, A. Silberschatz, HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads, PVLDB 2 (1) (2009).
[3] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, I. Stoica, Knowing when you're wrong: building fast and reliable approximate query processing systems, in: ACM SIGMOD, USA, 2014.
[4] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom, STREAM: The Stanford Data Stream Management System, Springer, 2004.
[5] B. Chandramouli, J. Goldstein, A. Quamar, Scalable progressive analytics on big data in the cloud, Proc. VLDB Endow. 6 (14) (2013).
[6] S. Chandrasekaran, M.J. Franklin, PSoup: a system for streaming queries over streaming data, VLDB J. 12 (2) (2003) 140–156.
[7] S. Chaudhuri, D. Das, U. Srivastava, Effective use of block-level sampling in statistics estimation, in: SIGMOD, 2004.

[8] S. Chaudhuri, G. Das, U. Srivastava, Effective use of block-level sampling in statistics estimation, in: SIGMOD, 2004.
[9] T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein, K. Elmeleegy, R. Sears, MapReduce online, in: Proc. of the 7th Conference on Networked Systems Design and Implementation, 2010.
[10] C. Cranor, T. Johnson, O. Spataschek, V. Shkapenyuk, Gigascope: a stream database for network applications, in: Proceedings of the ACM International Conference on Management of Data, SIGMOD, 2003.
[11] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Google research, http://research.google.com/archive/mapreduce.html.
[12] J. Dittrich, J.A. Quiane-Ruiz, A. Jindal, Y. Kargin, V. Setty, J. Schad, Hadoop++: making a yellow elephant run like a cheetah, PVLDB 3 (1) (2010).
[13] A. Doucet, M. Briers, S. Senecal, Efficient block sampling strategies for sequential Monte Carlo methods, J. Comput. Graph. Stat. (2006).
[14] A. Doucet, M. Briers, S. Senecal, Efficient block sampling strategies for sequential Monte Carlo methods, J. Comput. Graph. Stat. (2006).
[15] T.S. Ferguson, Optimal stopping and applications, Mathematics Department, UCLA, http://www.math.ucla.edu/~tom/Stopping/Contents.html, accessed March 2014.
[16] R.L. Grossman, Y. Guo, Parallel methods for scaling data mining algorithms to large data sets, in: Jan M. Zytkow (Ed.), Handbook on Data Mining and Knowledge Discovery, Oxford University Press, 2002, pp. 433–442.
[17] M.A. Hammad, T.M. Ghanem, W.G. Aref, A.K. Elmagarmid, M.F. Mokbel, Efficient pipelined execution of sliding-window queries over data streams, Technical Report TR CSD-03-035, Purdue University, Department of Computer Sciences, 2003.
[18] J.M. Hellerstein, R. Avnur, Informix under control: online query processing, Data Min. Knowl. Disc. (2000).
[19] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F.B. Cetin, S. Babu, Starfish: a self-tuning system for big data analytics, in: CIDR, 2011.
[20] S. Jaeger, Informational classifier fusion, in: Proc. of the 17th International Conference on Pattern Recognition, 2004.
[21] S. Jaeger, Using informational confidence values for classifier combination: an experiment with combined on-line/off-line Japanese character recognition, in: Proc. of the 9th International Workshop on Frontiers in Handwriting Recognition, 2004.
[22] C. Jermaine, S. Arumugam, A. Pol, A. Dobra, Scalable approximate query processing with the DBO engine, in: SIGMOD, 2007.
[23] D. Jiang, D.C. Ooi, L. Shi, S. Wu, The performance of MapReduce: an in-depth study, PVLDB 3 (1) (2010).
[24] D. Logothetis, K. Yocum, Ad-hoc data processing in the cloud, Proc. VLDB Endow. 1 (2) (2008) 1472–1475.
[25] M. Mokbel, X. Xiong, M. Hammad, W. Aref, Continuous query processing of spatio-temporal data streams in PLACE, Geoinformatics 9 (4) (2005).
[26] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G.S. Manku, C. Olston, J. Rosenstein, R. Varma, Query processing, approximation, and resource management in a data stream management system, in: Proceedings of the International Conference on Innovative Data Systems Research, CIDR, 2003.
[27] N. Pansare, V.R. Borkar, C. Jermaine, T. Condie, Online aggregation for large MapReduce jobs, in: PVLDB, 2011.
[28] G. Peskir, A. Shiryaev, Optimal Stopping and Free Boundary Problems, ETH Zuerich, Birkhäuser, 2006.
[29] V. Raman, B. Raman, J.M. Hellerstein, Online dynamic reordering for interactive data processing, in: VLDB, 1999.
[30] S. Reddy, K. Shilton, J. Burke, D. Estrin, M. Hansen, M. Srivastava, Evaluating participation and performance in participatory sensing, in: International Workshop on Urban, Community and Social Applications of Networked Sensing Systems, 2008.
[31] K. Singh, M. Xie, W.E. Strawderman, Confidence Distribution (CD) – distribution estimator of a parameter, in: Complex Datasets and Inverse Problems: Tomography, Networks and Beyond, vol. 54, 2007.
[32] S. Singh, N. Singh, Big data analytics, in: Proc. of the International Conference on Communication, Information and Computing Technology, 2012.
[33] B. Stigum, Econometrics and the Philosophy of Economics, Princeton University Press, 2003.
[34] Y. Yao, J. Gehrke, The cougar approach to in-network query processing in sensor networks, SIGMOD Rec. 31 (3) (2002).

Please cite this article in press as: K. Kolomvatsos et al., An Efficient Time Optimized Scheme for Progressive Analytics in Big Data, Big Data Research (2015), http://dx.doi.org/10.1016/j.bdr.2015.02.001