# Auto Regulated Data Provisioning Scheme with Adaptive Buffer Resilience Control on Federated Clouds

**Byungsang Kim**
Software Center, Samsung Electronics, 56, Seongchon-gil, Seocho-gu, Seoul, Korea
[e-mail: bs999.kim@samsung.com]

*Received August 9, 2016; revised October 12, 2016; accepted October 17, 2016; published November 30, 2016*

## *Abstract*

On large-scale data analysis platforms deployed on cloud infrastructures over the Internet, the instability of the data transfer time and the dynamics of the processing rate require a more sophisticated data distribution scheme which maximizes parallel efficiency by achieving the balanced load among participated computing elements and by eliminating the idle time of each computing element. In particular, under the constraints that have the real-time and limited data buffer (in-memory storage) are given, it needs more controllable mechanism to prevent both the overflow and the underflow of the finite buffer. In this paper, we propose an auto regulated data provisioning model based on receiver-driven data pull model. On this model, we provide a synchronized data replenishment mechanism that implicitly avoids the data buffer overflow as well as explicitly regulates the data buffer underflow by adequately adjusting the buffer resilience. To estimate the optimal size of buffer resilience, we exploits an adaptive buffer resilience control scheme that minimizes both data buffer space and idle time of the processing elements based on directly measured sample path analysis. The simulation results show that the proposed scheme provides allowable approximation compared to the numerical results. Also, it is suitably efficient to apply for such a dynamic environment that cannot postulate the stochastic characteristic for the data transfer time, the data processing rate, or even an environment where the fluctuation of the both is presented.

***Keywords:*** Federated clouds, in-memory computing, auto regulation, data provisioning model, buffer resilience control and optimization

A preliminary version of this paper appeared in IEEE Communication Letters in 2012

## 1. Introduction

As growing the Internet-scale federated cloud infrastucture, real-time distributed data processing platforms have gained momentum [1][2]. Especially, data intensive application such as high energy physics (HEP) experiments [3], online sequence alignment in Bioinformatics [4], Internet of Things [5] as well as real-time mobile cloud computing (MCC) [6][7] are centralizing the computing resources, services, data, and specific applications. Those devices produce unbounded measured dataset and it should be transfered to the autonomous storage and interacted with the target applications. Bulk data transfer protocols [17][19] leverage the efficient data sharing between remote sites. Also, the cloud pub/sub message middleware [8] or federated data synchronization platform [9] provide the solutions to share the data stream as near real-time.

Combining such the data to the Cloud computing platform, the balanced load distribution among the participating computing elements and the elimination of the idle computing elements are essential to minimize the total completion time (makespan) [10]. Especially, the in-memory computing paradigms that emphasize the ability to replace data in a finite memory buffer for high availability rather than constantly fetching them from slower storage, it requires a more sophisticated data scheduling scheme since the buffer capacity plays a major constraint on the parallel performance [11][12]. In particular, under conditions that the instability of data transfer time and the dynamics of data processing rate are presented, keeping an optimal level of data holdings in the buffer is a critical issue to minimize both memory space and the number of idle processes. During the last few decades, incorporation between data transfer time and data processing rate is well cultivated as a divisible load theory (DLT) paradigm [13][14]. The approaches have been based on centralized sender-driven data push models which focus on finding the optimal distribution ratio based on the closed-form formula or linear programming model under the deterministic assumptions that a sender (a master or a data source) has global and static knowledge about the data transfer time and the processing rate of all receivers (workers or computing sites).

However, current trend toward large scale distributed environment is characterized as dynamicity and elasticity that mean users are possible to compose their own resource pool by adding or dropping the computing elements. Especailly the dynamic resource provisioning such as auto-scaling features [15] or spot instance bidding concept [16] increase the necessity of the adaptable data provisioning mechanism in the aspects of the monetary or time-critical application. Such the collective computing model, the data processing rate presents stochastic behavior which depends on the number of computing elements, the capacity of the each computing element, and the granularity of the dataset. On the wide area network, moreover, it is more realistic that the data transfer time implies uncertainty which is introduced by sharing the link on geographically dispersed computing domains. For such large-scale computing model, the static assumptions of the sender-driven approaches are not able to tackle the practical issues as well as the robustness of the optimal solution does not be assured. Furthermore, under the limited buffer constraint is given, the complexity for finding the global solution is in fact NP-hard [14].

The principal motivation of our work is the desire to solve the problem based on decentralized receiver-driven data pull model. In particular, when the data transfer time and the data processing rate are fluctuated over time. As an opposition to the conventional centralized data distribution model, in this paper, we use the term – auto regulated data provisioning model on which, each receiver keeps an optimal level of reserved data in its finite data buffer and continuously replenishes the buffer with new data to prevent the buffer emptiness. So, the sender simply functions as a data server that responds to the data requests from the receivers. Such a decentralized data provisioning model makes it easy to manage the load distribution since each receiver can control the amount of the data in the buffer for itself by determining when and how much data should replenish. Our previous work about such the data provisioning service was shortly delivered in [26]. In this paper, we fully describe the system architecture of our data provisioning model as well as the sufficient analytical results with more elaborate mechanism of the proposed schemes. This paper makes the following contributions:

- We propose an auto regulated data provisioning model based on receiver-driven data pull mechanism. we place a data provisioning service (DPS) on each receiver side. The key feature of the service is to keep a certain amount of input data in the local data buffer by continuously replenishing the input data from the data source.
- We introduce the notion of buffer resilience and provisioning function which govern the overall performance of the data provisioning service. Based on these performance factors, we provide a synchronized data replenishment mechanism (SDRM) that makes it possible to implicitly avoid the data buffer overflow as well as explicitly regulate the buffer underflow by adequately adjusting the buffer resilience under the given provisioning function.
- We exploit an adaptive buffer resilience control (ABRC) scheme in order to find the optimal buffer resilience that minimizes both the buffer space and the waiting demands based on directly observed sample path analysis without any knowledge of the stochastic characteristics of the data replenishment time or the data processing rate.

The rest of this paper is organized as follows: section 2 provides research background and related works. Section 3 describes an auto regulated data provisioning model and its components. In section 4, we present the synchronized data provisioning mechanism and its analytic model. In section 5, we provide the adaptive buffer resilience control scheme including the criteria for finding optimal buffer resilience. Also, we validate the effectiveness of the proposed model and evaluate the performance with different scenarios in section 6. Finally, we summarize and conclude in section 7.

## 2. Background and Related Works

We consider large scale data analysis applications on geographically distributed computing domains with multiple remote data sources as shown in **Fig. 1**. The local data centers store huge size of divisible dataset that is contained in batch storage or real-time produced by internet connected devices in real-time. The computing elements such as private clouds or public clouds provide a big data platform including the parallel processing methods and user's application toolkit on a large set of physical or virtual machines(VM). So, the data sources should continuously provide a piece of input dataset among the computing domains as well as move the results back to the data sources.

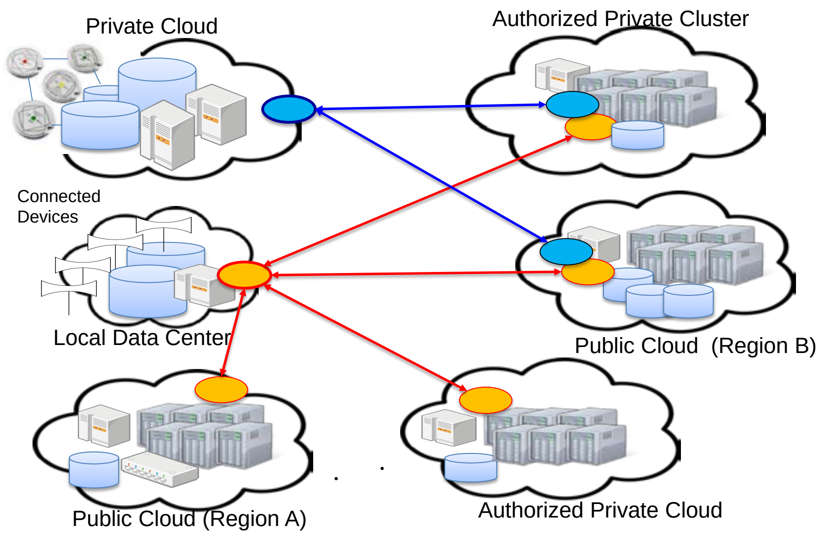


**Fig. 1.** Distributed data analysis environment with remote data sources

On the business domain, big data analytics is a rapidly growing field for processing huge size of in-house dataset [1][2][15]. The main approaches focus on batch processing paradigms using the distributed databases or the distributed file systems to provide almost linear scalability and fault tolerance. Recently, the memory computing based on resilience distributed data (RDD) concept [11][15] shows brilliant future of the real-time data analysis. However, they take little account of optimally importing or exporting the external data with on widely distributed area.

To enhance the end-to-end transfer performance, Fine-graned and Scalable TCP (FaST) [17], GridFTP [18] with striped transfer channels, on-demand secure circuits and advance reservation system (OSCARS) [19], or even the application-layer data throughput prediction and optimization service using multiple parallel TCP streams [20] are suggested. However, they focus on the improvement of end-to-end throughput without any considerations of storage constraints or real-time data processing rate on the target applications.

The balanced load distribution scheme incorporating the data transfer time and the data processing time is originated from divisible load theory [10] that divides the total load into the participating nodes based on closed-form optimization technique as a sender-driven data push model. Such data scheduling approaches have evolved with diverse assumptions for making it similar to real systems such as network topology, worker’s heterogeneity, number of scheduling rounds or communication mode [12]. However, these approaches assume a deterministic model that the sender has global and static knowledge about the data transfer time and the processing rate of all receiving nodes as well as the unlimited data buffer on the nodes. Improved approaches for overcoming the assumptions such as adaptive strategies [14] on resource unaware platform, or memory space constraints at the sink nodes [15] are suggested. However, the periodic scheduling nature in the sender-driven model makes it difficult to fully reflect the dynamic environment and the complexity to get the global solution is in fact NP-hard [14].

On the other hand, the receiver-driven data pull model is more practical and suitable for the unpredictable environments [20][21]. Pilot job systems such as DIANE [22] and DIRAC [23] are applied implementations. Such the data pull mechanism makes it possible to achieve the automatic load distribution since the receiver requests actively.

Furthermore, the complicated workflow jobs and their scheduling on federated clouds are cultivated well to apply on the practical applications. An autonomic management of the end-to-end execution for data-intensive application workflows in dynamic software-defined resource federation [23], a graph model that takes both QoS of Web services and QoS of network into a geographically distributed cloud datacenters [24], and a topology based workflow scheduling algorithm named Resource Auction Algorithm (REAL) [25] are proposed. However, those approaches do not touch the data provisioning scheme for reducing the data transfer time on the receivers. So, those inevitably introduce unnecessary data placement time. Consequently, in such a model, the volume of data, the location of sender, and the number of receivers become main causes to degrade the efficiency of the parallel throughput.

## 3. Auto Regulated Data Provisioning Model

In this section, we describe the proposed decentralized data provisioning model that has the data provisioning service in each receiver side. Under the assumption that a data object is a self-described and discrete processing unit which is a small part of the divisible dataset in an arbitrary data source, we describe the following components of the model in detail.

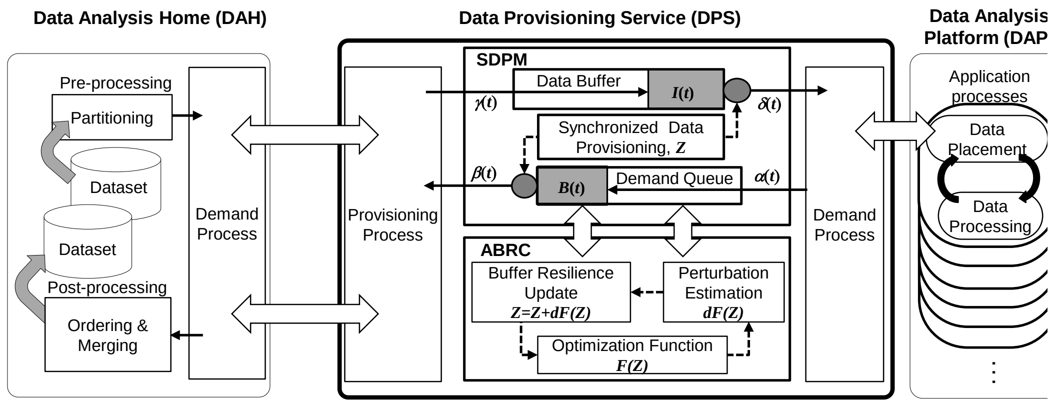
### 3.1 Architectural Model

***Data Analysis Home*** - On the data source, we deploy a *data analysis home* (DAH) for each dataset. The DAH is responsible for distributing the input data objects and collecting the output data objects. As shown in **Fig. 2(a)**, it has a preprocessing stage for partitioning the dataset into small input data objects and a post-processing stage for ordering and merging the output data objects into permanent storage. In addition, the DAH has a demand process which simply functions as a data server for responding to the requests from data provisioning services. When a request comes to the demand process, the DAH unloads an output data object (if has) and responds with a new data object to the requester.

***Data Analysis Platform –*** In the computing domain, a *data analysis platform* (DAP) is composed of a set of independent application processes for processing the data objects on multiple computing nodes. As shown in the right part of the **Fig. 2(b)**, each application process has an infinite loop procedure - data placement process to locate the input data object and a data processing process to produce the output data object repeatedly. Since the data placement state is on the idle state of the process, it should be eliminated or regulated.

***Data Provisioning Service*** - On each computing domain, we deploy a data provisioning service (DPS) - the core part of the model. The DPS is responsible for the data provisioning process from the DAH to the DAP. So, the DPS should respond the request from the application processes as well as collect the output data objects. On the other hand, the DPS should replenish the input data object from the DAH as well as return the collected results. To manage such application data, the DPS has a finite input buffer and a demand queue as shown in the **Fig. 2(b)**. The input buffer stores input data object and forwards them to the application process. Also, the requests from the DAP is managed by the demand queue.

In order to guarantee efficient operation, the DPS should determine how many data objects should be kept in the data buffer and when it should replenish the input data objects and return



(a) DAH in Remote Data source (b) DPS and DAP in Local Computing Domain

**Fig. 2.** The proposed system architecture of the data provisioning model

the output data object from/to DAH. Since the data buffer is finite, determining those performance factors is essential to preventing the underflow (in that case, the DAP introduces idle processes) or the overflow (in that case, the input data object is blocked) of the data buffer.

In the following section, we present the operation mechanism of the DPS for regulating such unwanted states.

## 3.2 Synchronized Data Replenishment Mechanism

In this section, we provide the operation mechanism of the DPS with the analytic model. In particular, the synchronized data replenishment mechanism (SDRM) for managing the data buffer and the demand queue is described in detail. Prior to analyzing the behaviors of the DPS, we premise the following assumptions: (A1) We represent a data object as a discrete processing unit which is a small part of divisible input dataset. So, the occupancy of the data buffer means the number of data objects in the buffer. (A2) We assume the output data object is relatively smaller compared to the input data object. So, the output data object is piggybacked to the demand requests. (A3) The data replenishment time includes the transfer time and the waiting time for obtaining a single data object from the remote DAH. We assume that the data transfer time is predominant from remote DAH to the DPS but negligible within local domain (from the DPS to the DAP). So, the waiting processes in DAP is occurred only when the data buffer is empty. Furthermore, we define the notations of the main and auxiliary parameters on the **Table 1**. Based on the assumptions and definitions, we model an arbitrary DPS as a discrete event system which is combined by the following two stochastic processes:

***(i)*** ***Data demand process*** - When a request event arrives in the demand queue, the DPS responds with a data object in the data buffer. If the buffer is empty, the event should wait in the demand queue until the buffer has been filled. As shown in **Fig. 2(b)**, we denote the rate notations of the number of arrival and departure processes in the demand queue during (0, $t$) as $\alpha(t)$ and $\beta(t)$ receptively.

***(ii)*** ***Data provisioning process*** - When the DPS sends a request event to the DAH, a new input data object comes to the data buffer after elapsing the replenishment time. As shown in **Fig. 2(b)**, we denote the rate notations of the number of arrival and departure processes in the data buffer during (0, t) as $\gamma(t)$ and $\alpha(t)$ receptively.

**Table 1.** Definition of the main and auxiliary parameters and their notations

| Notations | Definitions |
|---|---|
| $Z$ | The buffer resilience that represents the reserved number of data objects. |
| $I(t)$ and $E_I(z)$ | The number of stored data objects in the data buffer at $t$ and its average occupancy when $Z=z$ |
| $B(t)$ and $E_B(z)$ | The number of demand requests in the demand queue at $t$ and its average occupancy when $Z=z$ |
| $N(t)$ | The provisioning function at $t$ |
| $PPR$ | The processing to provisioning ratio |
| $F(z)$ | The provision cost function |
| $Z^*$ | The optimal buffer resilience |
| $F_n(z^*)$ | The optimal (minimum) provisioning cost function on the iteration $n$ |
| $\upsilon$ | The weight factor between data buffer and the demand queue which makes trade-off between memory cost and the waiting time penalty |
| $\nu$ | The step size in every iteration |

To avoid data buffer overflow, the DPS synchronizes aforementioned two processes - (*i*) and (*ii*), e.g., as soon as a demand process is completed, the DPS triggers a replenishment process. So, after elapsing the replenishment time, the number of data objects in the data buffer is restored the same as the previous state. Note that the DPS will never be operated if the data buffer is initially empty. So, we introduce buffer resilience ($Z$) which represents the reserved number of data objects. Before starting the SDRM, the DPS determines the size of the buffer resilience, $Z$, and prepares data objects as much as $Z$ using replenishment process.

Let $I(t)$ and $B(t)$ be the number of stored data objects in the data buffer and the number of demand requests in the demand queue at time $t$ respectively. Then, $I(0) = Z$ and $B(0) = 0$ by the initial condition, as well, by the arrival and departure processes of the data buffer and the demand queue, we can identify the occupancies of those at time $t$ as $I(t)= Z+\gamma(t)-\delta(t)$ and $B(t)= \alpha(t)-\beta(t)$, (see **Fig. 2(b)**). On our synchronized mechanism, since the two departure processes occur at same time, the $\beta(t)$ and the $\alpha(t)$are identical. Thus, we get the following relation:

$$I(t) - B(t) = Z + \gamma(t) - \delta(t) - \alpha(t) + \beta(t) = Z + \gamma(t) - \alpha(t). \tag{1}$$

$I(t) > 0$ implies $B(t) = 0$, (if the data buffer is not empty, the demand queue always empty), on the other hand, $B(t) > 0$ implies $I(t) = 0$ (the demand requests should stay in the demand queue when the data buffer is empty). As well, both are non-negative integer, we obtain $I(t)$ and $B(t)$ from (1) as

$$I(t) = \max[Z - N(t), 0], \tag{2}$$

$$B(t) = \max[N(t) - Z, 0], \tag{3}$$

where we denote $N(t)$ as the provisioning function of the DPS, which is defined by aforementioned two arrival rates:

$$N(t) = \alpha(t) - \gamma(t),\ \alpha(t) > \gamma(t), t > 0. \tag{4}$$

The provisioning function $N(t)$ and aforementioned buffer resilience $Z$ are important parameters to govern overall performance of the DPS. Furthermore, we can identify that the $N(t)$ is represented as a general queuing system of which arrival rate is data demand rate and the service time is the replenishment time, since the arrival rate $\alpha(t)$ and the departure rate $\gamma(t)$ of the $N(t)$ are defined by random variables which are correspondent to the data demand rate and the replenishment rate of the DPS. So, for the $N(t)$, we define the processing to provisioning ratio (PPR) which is represented by

$$PPR = \frac{\gamma(t)}{\mu(t)},\ \gamma(t) > 0, \mu(t) > 0, t > 0, \tag{5}$$

where the $\gamma(t)$ means the average demand rate from DAP and the $\mu(t)$ is the maximum data replenishment rate from DAH during $(0, t)$. The PPR is similar to the CCR (computation to communication ratio) [14] which is a qualitative measure widely used on distributed applications but it is different in that the PPR is characterized by the stochastic variables. On the other hand, the buffer resilience $Z$ can be used as the qualitative index of the maximum buffer size for regulating both the occupancy of the data buffer and the demand queue.

Consequently, under the given $N(t)$, the proposed SDRM makes it possible to automatically avoid data buffer overflow by determining the buffer size as much as the buffer resilience since the $I(t)$ is bounded to the $Z$ as shown in (2). In addition, it is possible to explicitly regulate $B(t)$ by adjusting $Z$ since $B(t)$ appears when $N(t) > Z$ as shown in (3). So, the remaining problem is how to determine the optimal buffer resilience. We provide the methodology with cost minimized function for the optimality criteria in the following section.

## 4. Adaptive Buffer Resilience Control Scheme

As shown in previous results, $I(t)$ and $B(t)$ are only dependent on the provisioning function $N(t)$ and the buffer resilience $Z$. So, it is possible to provide the designated quality of service by adjusting the buffer resilience under given $N(t)$. In this section, firstly, we define the provisioning cost function and the criteria for finding optimal buffer resilience. Secondly, we develop an adaptive buffer resilience control (ABRC) algorithm which can find directly the optimal buffer resilience on the sample path of the occupancy of the data buffer and the demand queue under the $N(t)$ is unknown.

### 4.1 Provisioning Cost Model and Optimality Criteria

In steady state, the $N(t)$ can be modeled as a general queuing system of which the average data demand rate is $\gamma$ and the maximum data replenishment rate is $\mu$. So, the stability condition of the data buffer is $PPR = \gamma/\mu < 1$.

On the stability condition, the physical meaning of $I(t)$ is the memory space for storing the data objects at time $t$. Supposing $E_I(z)$ is the average occupancy of the data buffer when $Z = z$, it can be represented by the average memory cost for reserving the application data. As shown in (2), the smaller $z$ is given, the smaller $I(t)$ is guaranteed since the $N(t)$ is independent from $z$. So, we should keep the buffer resilience as small as possible to minimize the average memory cost. On the other hand, the $B(t)$ reflects the number of idle processes which do not participate in the data processing process. Denoting $E_B(z)$ as the average occupancy of the demand queue when $Z = z$, the larger $z$ is given, the smaller $B(t)$ is guaranteed as (3). So we should keep the buffer resilience as large as possible to minimize the waiting time of the demands. Since the two criteria have negative relationship in terms of $Z$, we formulate *provisioning cost function*, $F(z)$ that is composed of a weighted sum of the two criteria as

$$F(z) = (1-\upsilon)E_I(z) + \upsilon E_B(z),\ 0<\upsilon<1, \tag{6}$$

where the $\upsilon$ is a weighted factor to the demand queue. So, by customizing the $\upsilon$, users can make trade-off between memory cost and the waiting time penalty. Consequently, we can identify that the *optimal buffer resilience*, $Z*$ is the size of the buffer resilience that minimizes the provisioning cost function, $F(Z* = z)$. It is determined by $dF(z)/dz = 0$ since the $F(z)$ is convex to $z$ in positive area.

For example, supposing the provisioning function, $N(t)$ follows M/M/1 queuing model which means that there is a single server (DAH), the demands follows the Poisson arrival pattern, and the data replenishment time is exponentially distributed. Denoting the probability distribution function (PDF) of the $N(t)$ is $P(N = n)$, then, the PDF of both $\{I(t), t > 0\}$ and $\{B(t), t > 0\}$is given by

$$P(I=n) = \begin{cases} P(N>Z), & n=0 \\ P(N=Z-n), & n=1,2,\dots,Z, \end{cases} \tag{7}$$

$$P(B=n) = \begin{cases} P(N>Z), & n=0 \\ P(N=Z+n), & n=1,2,\dots,Z. \end{cases} \tag{8}$$

Denoting $\rho$ as the PPR of the $N(t)$, the state probability of $N(t)$ is $P_N(n)= (1-\rho)\,\rho^n$, $n> 0$. Then, from (7) and (8), the state probabilities of $I(t)$ and $B(t)$ are represented by $P_I(n) = P_N(Z-n)= (1-\rho)\,\rho^{Z-n}$, $0 < n> Z$ and $P_B(n)= P_N(n+Z)= (1-\rho)\,\rho^{n+Z}$, $n > 0$ respectively. Hence, the *provisioning cost function* defined by (6) is

$$\begin{aligned} F(z)_{MM1} &= (1-\upsilon)E_I(z) + \upsilon E_B(z) \\ &= (1-\upsilon)\left(\textstyle\sum_{n=1}^{Z}(1-\rho)\rho^{Z-n}\right) + \upsilon\left(\textstyle\sum_{n=1}^{\infty}(1-\rho)\rho^{n+Z}\right) \\ &= (1-\upsilon)\left(z - \frac{\rho(1-\rho^z)}{1-\rho}\right) + \upsilon\left(\frac{\rho^{Z+1}}{1-\rho}\right). \end{aligned} \tag{9}$$

By calculating $(dF(z)_{M/M/1}/dz)= 0$, we can obtain the optimal buffer resilience $Z^*$ that minimize the provisioning cost function as

$$Z^* = \frac{\ln(1-\upsilon)}{\ln(\rho)}. \tag{10}$$

### 4.2 Adaptive Buffer Resilience Control Algorithm

Assuming that the stochastic behavior of $N(t)$ is known and stationary, we can obtain the minimum provisioning cost and the optimal buffer resilience by $dF(z)/dz = 0$ since the $F(z)$ is convex to $z$ in positive area. In practical, however, it is difficult to identify the stochastic behavior of $N(t)$ as well as the notion of the steady state is hard to justify under the fluctuation of both data demand rate and data replenishment time. So, a more practical scheme is necessary. We develop an adaptive buffer resilience control (ABRC) algorithm by estimating $dF(z_n)/dz$ based on perturbation analysis of directly observed sample path over $n^{th}$ finite time area. We can then obtain $(n + 1)^{th}$ optimal buffer resilience $z_{n+1}$ through an iterative form on [27]

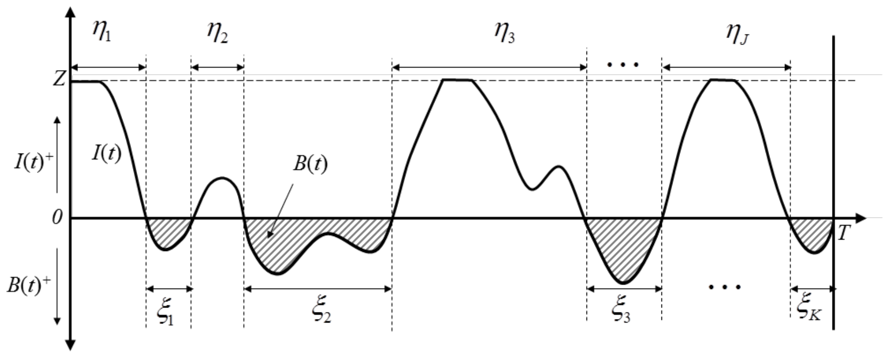


**Fig. 3.** A generic sample path on the data buffer and the demand queue during (0, T).

$$Z^*_{n+1} = Z^*_n - \nu f(Z^*_{n+1}), n = 0,1, \dots \tag{11}$$

where $\nu$ is the step size and $f_n(z^*)$ is an estimate of $dF(z^*_n)/dz$ for $n^{th}$ iterations which is calculated by

$$f(Z^*_n) = \frac{dF(Z^*_n)}{dz} = (1 - \upsilon)\left(\frac{dE_I(Z^*_n)}{dz}\right) + \upsilon\left(\frac{dB_I(Z^*_n)}{dz}\right). \tag{12}$$

To obtain the $f(Z^*_n)$, we apply the results of [30], in which work the authors derived the infinitesimal perturbation analysis (IPA) estimate of the occupancy of a finite buffer with respect to buffer size by approximating the G/G/1 finite queuing system as stochastic fluid model. The estimate turned out to be the sum of all intervals of the surplus periods of the buffer since the arrival and departure process are independent from the buffer size.

In our model, supposing the $N(t)$ follows G/G/1 queuing system, the $E_I(Z^*_n)$ and the $E_B(Z^*_n)$ are the same results since the $N(t)$ is independent from $z$. So, the derivative of $E_I(Z^*_n)$ is approximated by the sum of all intervals of the $I(t) > 0$ periods in each iteration as

$$\frac{dE_I(Z^*_n)}{dz} = \frac{1}{T}\sum_{j=0}^{J} \frac{d}{dz}\int (Z - N(t))dt = \frac{1}{T}\sum_{j=0}^{J} \eta_j \,. \tag{13}$$

where $\eta_j$ is the $j^{th}$ surplus period of the data buffer during (0, $T$), $0>j> J$, and $\eta_j \leq T$ (see Fig.3). Similarly, the derivative of $E_B(Z^*_n)$ with respect to $z$ over (0, $T$) is

$$\frac{dE_B(Z^*_n)}{dz} = \frac{1}{T}\sum_{k=0}^{K} \frac{d}{dz}\int (N(t) - Z)dt = \frac{1}{T}\sum_{k=0}^{K} \xi_k \,. \tag{14}$$

where $\xi_k$ is the $k^{th}$ surplus period of the demand queue over (0, $T$), $0>k>K$, and $\xi_{jk} \leq T$ (see Fig. 3). Hence, the estimated derivative of the provisioning cost function over [0,$T$] yields

$$f(Z^*_n) = \frac{1}{T}\left((1-\upsilon)\sum_{j=0}^{J}\eta_j - \upsilon\sum_{k=0}^{K}\xi_k\right). \tag{15}$$

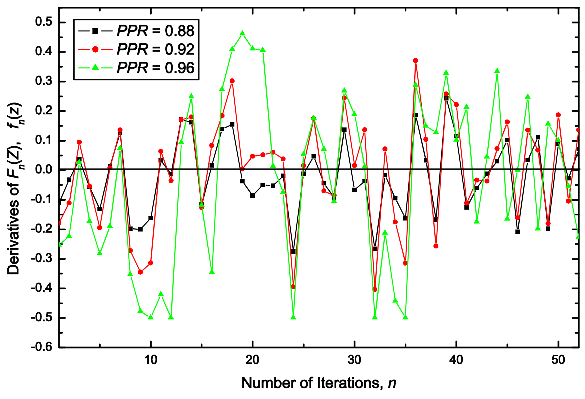
**Algorithm 1.** Adaptive Buffer Resilience Control (ABRC)

```
1:  Set $ABRC_{INIT}=(Z_0, \nu, T)$ and $\upsilon$
2:  $Z_{previous} = Z_0$
3:  while DPS is running do
4:    $t_i = 0, t_b = 0, \tau = 0, T_n = t + T,\ Z_{previous} = Z_{current}$
5:    while $t < T_i$ do
6:      if $I(t) = 0$ then
7:        $t_i = t_i + (t - \tau)$
8:      else if $B(t) = 0$ then
9:        $t_b = t_b + (t - \tau)$
10:     else if $I(t) > 0$ then
11:       $\tau = t$
12:     else if $B(t) > 0$ then
13:       $\tau = t$
14:     end if
15:   end while
16:   $f = \{(1 - \upsilon)t_i - (\upsilon) \times t_b\}/T$
17:   $Z_{current} = Z_{previous} - \lceil \nu \times f \rceil$
18:   if $Z_{current} > Z_{previous}$ then
19:     Increase current buffer resilience to $Z_{current}$
20:   else if $Z_{current} < Z_{previous}$ then
21:     Decrease current buffer resilience to $Z_{current}$
22:   else
23:     Keep current buffer resilience
24:   end if
25: end while
```

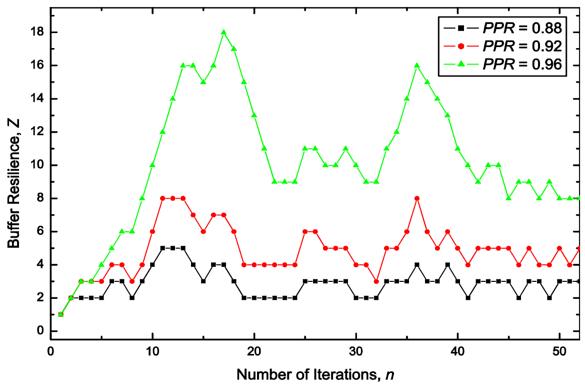
The result of (15) implies the simplicity of the ABRC scheme which only identifies the empty states of the data buffer and the demand queue in order to update the buffer resilience for each iteration. The pseudo expression of the ABRC scheme is shown in Algorithm 1. On the initialization (line [1– 2]), the ABRC parameters, $ABRC_{INIT}$=(*initial buffer resilience*, *step size*, *measured interval*) and the weighted factor are configured. Then the DPS can the derivatives of the data buffer and the demand queue ($t_i$, $t_b$) by observing the empty epochs to add up the surplus periods until the current time $t$ reaches the measured interval $T$ (line number [*5 – 15*]). At the end of the interval (line [16]), the estimate of the provisioning cost, $f_n(z)$ yields and finally the current buffer resilience, $Z_{current}$ is updated by comparing the previous buffer resilience $Z_{previous}$ in the line [17 -23].

## 5. Performance Evaluation

In this section, we present our simulation platform and the evaluations of the proposed model. We utilize SimJava [29] modeling package as a discrete event simulation tool. The evaluations are composed of three parts. First, we verify the effectiveness of the ABRC scheme by comparing with numerical results. Second, we compare the ABRC scheme with static buffer resilience scheme which does not apply the adaptive method. Finally we show the applicability of the ABRC scheme in the general distributed, unsteady and fluctuated environment of demand arrival rate and the replenishment time.



(a) Estimates of the $f(Z^*_n)$, $v$=5, $T$=750

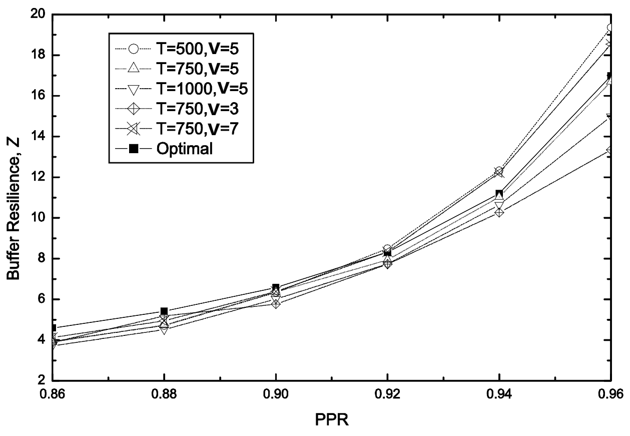


(b) Buffer resilience on each iteration,$v$=5, $T$=750

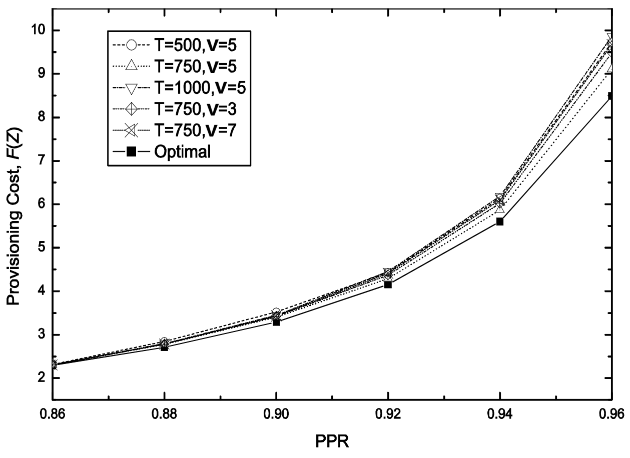
**Fig. 4.** Estimates of the derivative of the provisioning cost and the adjusted buffer resilience

## 5.1. Verification of ABRC scheme

Under the provisioning function $N(t)$ follows the M/M/1 queuing model, we firstly examine the $f(z)$and the $Z$ on three different PPRs as 0.88, 0.92 and 0.96. The initial parameters of ABRC are configured as $\upsilon$ = 0.5 and the $ABRC_{INIT}$ is $Z_0$ = 1, the step size $\nu$ = 5, and the measured interval on each iteration $T$ = every 750 arrivals of the demands. We performed total 52 iterations. **Fig. 4(a)** shows the estimates of the derivative of the provisioning cost on each iteration. When the value is negative, the buffer resilience of the next iteration is increased as much as the integer value of the product of the estimate and the scale size, e.g. $\lceil \nu f_n^* \rceil$. In contrary, if the value is positive, the buffer resilience is decrease as much as $\lceil \nu f_n^* \rceil$. It corresponds to the line [17 -23] in **Algorithm 1**. On the plot, the fluctuation of the estimates becomes large when the PPR is increased. As the iterations are repeated, however, we are able to observe that the range of the fluctuation decrease. These results reflect the change of the buffer resilience in **Fig. 4(b)**. We can observe that each trial finds the proper level of the buffer resilience as the iteration is repeated even though the initial buffer resiliencies are fixed to one. Furthermore, those flexibly adjust the buffer resilience in according to the current density of the provisioning function on each iteration.



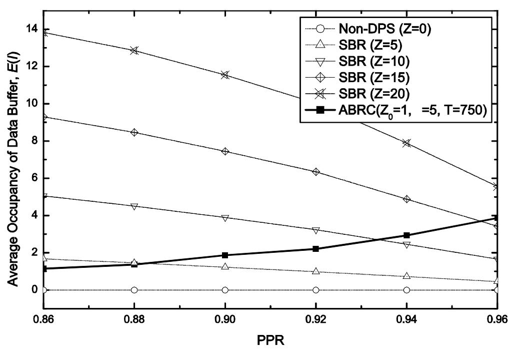
(a) Average buffer resilience



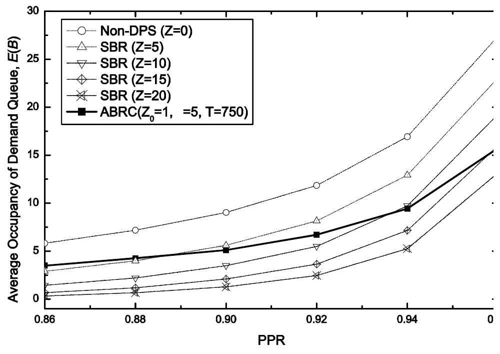
(b) Average provisioning cost

**Fig. 5.** Comparison of the buffer resiliencies between the numerical results and the ABRC

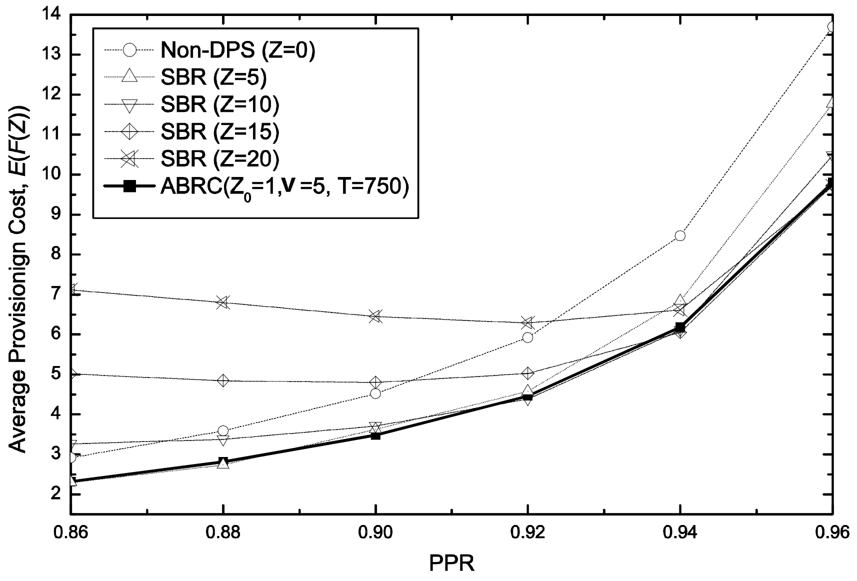
In order to compare to the numerical results, we conduct more experiments with different initial configurations as $v$ = 3, 5, 7 and $T$ = 500, 750, 1000 on six different PPRs. **Fig. 5** shows the average buffer resiliencies and the average provisioning cost. In the plots, the optimal values were calculated by (9) and (10). As shown in **Fig. 5(a)**, the average buffer resilience of the most trials are placed near to the optimal values (black squares) regardless of the initial configurations. However, when the PPRs are closed to 1, the gap to the optimal value tends to be large in every trial. When compared to the optimal values, in the case the measured interval is small ($T$ = 500) or step size is large ($v$ = 7), the buffer resilience is over-estimated, in contrary, when the measured interval is large ($T$ = 1000) or step size is small ($v$ = 3), the buffer resilience is under-estimated. On the same configuration, **Fig. 5(b)** shows the average provisioning cost of the trials. When the PPRs are increased, the gap to the optimal value is more observed and the values are larger than those of the optimal ones. These errors are caused by the frequent appearance of the bursty demand and the replenishment rate on high PPR. The error appears much more since the sensitivity of the burst traffic increase when the step size is large or the measured interval is small.



(a) Average occupancy of the data buffer



(b) Average occupancy of the demand queue



(c) Average provisioning cost

**Fig. 6.** Comparison of the average occupancy of the data buffer and the demand queue, and the average provisioning cost among Non-DPS, SBRs, and proposed ABRC

## 5.2. Comparison of the static buffer resilience scheme

In this scenario, we compare the average occupancies of the data buffer and the demand queue, and the average provisioning cost on different configurations - cases as the DPS with ABRC scheme (*proposed ABRC*), the DPS with static buffer resilience (*SBR*), and a case without DPS itself (*non-DPS*). In the SBR cases, the buffer resilience is fixed to the initial value. We set four different initial buffer resiliencies as SBR($Z_0$ = 5), SBR($Z_0$ = 10), SBR($Z_0$ = 15), and SBR($Z_0$ = 20). In the case of the non-DPS, the buffer resilience is zero ($Z_0$ = 0) since it has no space to store the data objects. For the initial parameters of the proposed ABRC, we set υ= 0.5 and the $ABRC_{INIT}$ = ($Z_0$ = 1, $v$ = 5, $T$ = 750).

**Fig. 6(a)** compares the average occupancy of the data buffer on six different PPRs. In case of the non-DPS, the average occupancy is always zero since it does not use the data buffer. SBR($Z_0$ = 10), SBR($Z_0$ = 15), and SBR($Z_0$ = 20). In the case of the non-DPS, the buffer resilience is zero ($Z_0$ = 0) since it has no space to store the data objects. For the initial parameters of the proposed ABRC, we set υ= 0.5 and the $ABRC_{INIT}$ = ($Z_0$ = 1, υ= 5, $T$ = 750). **Fig. 6(a)** compares the average occupancy of the data buffer on six different PPRs. In case of the non-DPS, the average occupancy is always zero since it does not use the data buffer. On the other hand, those of the SBRs are decreased proportionally when the PPR is increased. It occurs since the data buffer needs restoration time to recover the initial state. When the PPR is high, the average occupancy is more decreased since the restoration time takes longer than that of the small PPR. On contrary, the proposed ABRC case increases the average occupancy of the data buffer when the PPR is increased. It is because that the ABRC scheme adjusts the buffer resilience in order to minimize the provisioning cost function on each PPR. Intuitively, we can identify that the SBRs belong to the over-provisioned state if the average occupancies are larger than those of the ABRC case, otherwise, those are in the under-provisioned state.

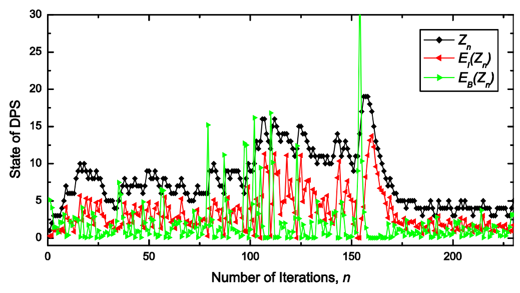
The average occupancy of the demand queue in **Fig. 6(b)** shows the inverse phenomenon to the data buffer. In case of the non-DPS, the average occupancy is always larger than others in all PPRs. On the other hand, those of the SBRs are increased proportionally when the PPR is increased. It occurs since the empty probability of the data buffer is increased when the PPR is high. In the case of the ABRC, the average occupancy is increased to the PPRs. Contrary to the state of the data buffer, the SBRs belong to the over-provisioned state if the average occupancies are smaller than that of the ABRC case, otherwise, those are in the under-provisioned state. Using the **Fig. 6(c)**, we can synthetically explain the above observations. The plots show the average provisioning costs of the trials. The proposed ABRC case keeps the smallest values over all PPRs. Meanwhile, the SBRs are close to the minimum cost only on specific points according to their initial buffer resiliencies such as $Z_0$=5 at (0.86, 0.88, 0.90), $Z_0$=10 at (0.92, 0.94), $Z_0$=15 at (0.94, 0.96), and $Z_0$=20 at 0.96, respectively. On the non-DPS case, it cannot achieve the minimum costs at all PPRs. To sum up, by minimizing the average provisioning cost, the proposed ABRC scheme enables to guarantee the optimal buffer resilience that synthetically satisfies both the data space and the waiting demands during run-time.

## 5.3. Adaptivity of the ABRC scheme

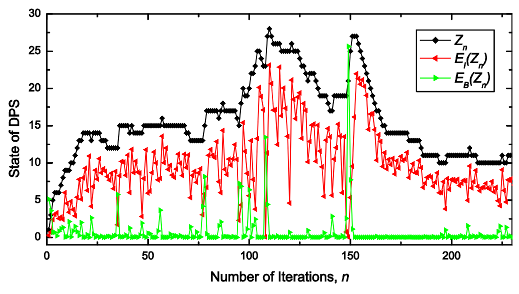
In this scenario, we examine the adaptability of the ABRC scheme on the condition that the arrival of demands fluctuates on run-time as well as the replenishment time has different variations. In addition, the weighted factors are customized. We prepared three different cases (*case*-1, *case*-2, and *case*-3) as shown in **Table 2**. The demand arrival rate is divided to three stages (*s*1, *s*2, and *s*3) where the *s*2 occurs from 100 to 150 iterations. In addition, the *case*-1

**Table 2.** Parameters for the three cases, where we denote *uni*(*a*, *b*) as uniform distribution with *mean* = *a* and *variation* = *b* and *exp*(*a*) as exponential distribution with *mean* = *a*

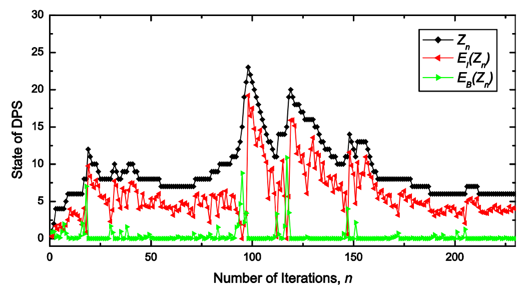
| Cases | *Stage* | $\upsilon$ | $1/\lambda$ | $1/\mu$ |
|---|---|---|---|---|
| | *s*1 | | *uni(10,10)* | |
| *Case*-1 | *s*2 | *0.7* | *uni(9.5,9.5)* | *Exp(9)* |
| | *s*3 | | *uni(10.5, 10.5)* | |
| | *s*1 | | *uni(10,10)* | |
| *Case*-2 | *s*2 | *0.9* | *uni(9.5,9.5)* | *Exp(9)* |
| | *s*3 | | *uni(10.5, 10.5)* | |
| | *s*1 | | *uni(10,10)* | |
| *Case*-3 | *s*2 | *0.9* | *uni(9.5,9.5)* | *Uni(9,9)* |
| | *s*3 | | *uni(10.5, 10.5)* | |



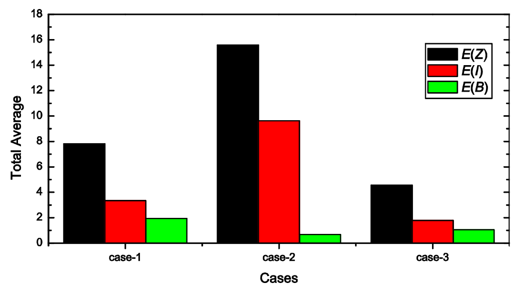
(a) $Z_n$ , $E_I(Z_n)$, $E_B(Z_n)$ on each iteration under the case-1



(b) $Z_n$ , $E_I(Z_n)$, $E_B(Z_n)$ on each iteration under the case-2



(c) $Z_n$ , $E_I(Z_n)$, $E_B(Z_n)$ on each iteration under the case-3



(d) Total average of the cases

**Fig. 7.** Comparison of the states of $Z_n$, $E_I(Z_n)$, $E_B(Z_n)$ on each iteration.

has smaller weighted factor (υ= 0.7) compared to the *case*-2 (υ= 0.9). On the other hand, *case*-2 has the larger variation of the replenishment time (*exp*(9)) compared to *case*-3(*uni*(9, 9)). **Fig. 7** shows the results of the average size of the buffer resilience and the average occupancies of the data buffer and the demand queue on each iteration. About all cases, the magnitude of the buffer resilience on s2 is larger than those of the s1 and s3 since the demand arrival interval is decreased. On the other hand, **Fig. 7(a)** and **Fig. 7(b)** show the effect of the weighted factor. Even though those have the same demand arrival and replenishment time, the buffer resilience of the *case*-2 is larger than that of the case-1 since the weighted factor of the case-2 is larger than that of the *case*-1. It leads to the smaller demand occupancy of the *case*-2 compared to the case-1 whereas the larger data buffer occupancy. In addition, **Fig. 7(b)** and **Fig. 7(c)** present the effect of the variation of the replenishment time. The buffer resilience of the *case*-3 is much smaller than that of the *case*-2, even though those have the same demand arrival rate and weighted factor, since the variation of the replenishment time of the case-3 is smaller than that of the *case*-2. Thus, the case-3 enables to achieve the similar demand occupancy with much less buffer resilience compared to the *case*-2. Total average of the buffer resilience and the occupancies are compared in **Fig. 7(d)**. The results show that proposed ABRC scheme is possible to adaptively control the buffer resilience without any knowledge of the stochastic characteristics of the data replenishment time and the data processing rate. Furthermore, the ABRC scheme gives user the opportunity to enhance the service quality (reducing the waiting time of the application processes) by undertaking the data buffer usage cost.

## 7. Conclusion

This paper presents a decentralized data provisioning model which enables the auto regulated load balancing as well as the reduction of the data transfer time in terms of the receivers. Under the limited buffer capacity and unpredictable situation of demand rate and the replenishment time, the proposed synchronized data replenishment mechanism (SDRM) makes it possible to implicitly avoid the data buffer overflow as well as to regulate the buffer underflow by adequately adjusting the buffer resilience. To find the optimal buffer resilience even under such unsteady environment, the adaptive buffer resilience control (ABRC) scheme is exploited, which minimizes both the buffer space and the waiting demands based on directly observed sample path analysis without any knowledge of the stochastic characteristics of the provisioning function. The evaluations verify that the proposed ABRC scheme shows good approximation compared to the numerical results as well as it seeks the optimal buffer resilience automomously even though the specific probability law of the provisioning function is not postulated. Furthermore, the ABRC scheme gives user the opportunity to enhance the service quality by undertaking the data buffer usage cost. In the paper, we assumed that a single data provisioning server (DPS) which are able to be supported when the processing to provision rate (PPR) is less than 1. If the provisioning rate can not keep up with the processing time, (e.g, highly data intensive with parallel application), we need to consider a scalable DPS architecture with clustered provisioning servers on the single data center. Even the case, we think our adaptive resilience control scheme will work properly since each of the DPS operates as a autonomously regulated manner. Another restriction on the proposed scheme is that, the demand and the replenishment size are a single type and provided by the linear estimation. We will do further work for the batch patterns of the demand size, the replenishment size, or both. Such the batch replenishment scheme will reduce the replenishment time under the condition that the additional request cost (setup time) is imposed. In that case, finding the optimal batch size is another issue to enhance the performance of the data provisioning service. We are interested in the nonlinear estimation techniques to cope with such kinds of the bursty shape of the provisioning function and to find optimal batch size as well as the optimal buffer resilience.

## References

[1] A. Jacobs, The pathologies of big data, *Commun. ACM,* vol. 52, pp. 36–44, 2009. Article (CrossRef Link).

[2] J. Dean, S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM* 51, pp. 107–113, 2008. Article (CrossRef Link).

[3] J. Andreeva, S. Campana, F. Fanzago, J. Herrala, "High-energy physics on the grid: the atlas and cms experience," *Journal of Grid Computing*, vol. 6, no. 1, pp. 3–13, 2008. Article (CrossRef Link).

[4] Erlich, Yaniv, "A vision for ubiquitous sequencing," *Genome Research*, vol. 25, no. 10, pp 1411–1416, 2015. Article (CrossRef Link).

[5] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, "From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence," *Elsevier*, 2014. Article (CrossRef Link).

[6] H. Qi and A. Gani, "Research on mobile cloud computing: Review, trend and perspectives," *Second International Conference on Digital Information and Communication Technology and it's Applications*, pp. 195-202, 2012. Article (CrossRef Link).

[7] Khan, A.N., Kiah, M.L.M., Ali, M. et al., "BSS: block-based sharing scheme for secure data storage services in mobile cloud environment," *Journal of Supercomputing*, vol. 70, no 2, pp 946-976, 2014. Article (CrossRef Link).

[8] J. Gascon-Samson, F. P. Garcia, B. Kemme and J. Kienzle, "Dynamoth: A Scalable Pub/Sub Middleware for Latency-Constrained Applications in the Cloud," in *Proc. of IEEE 35th International Conference on Distributed Computing Systems (ICDCS)*, pp. 486-496, 2015. doi: 10.1109/ICDCS.2015.56. Article (CrossRef Link).

[9] D. Huang, A. Jaikar, G. Kim, Y. Kim, and S. Noh, "A Self Synchronization Mechanism in a Federated Cloud," *International Journal of Software Engineering and Its Applications*, vol. 10, no. 1, pp. 233-240, 2016. Article (CrossRef Link).

[10] D. G. O. Veeravalli Bharadwaj, Thomas G. Robertazzi, "Scheduling Divisible Loads in Parallel and Distributed Systems, "*IEEE Computer Society Press*, 1996. Article (CrossRef Link).

[11] Y.Wang, H. Chen, B.Wang, J. M. Xu, H. Lei, "A scalable queuing service based on an in-memory data grid," in *Proc. of 2010 IEEE 7th International Conference on e-Business Engineering (ICEBE)*, pp. 236 –243, 2010. Article (CrossRef Link).

[12] B. Veeravalli, J. Yao, "Divisible load scheduling strategies on distributed multi-level tree networks with communication delays and buffer constraints," *Computer Communications*, vol. 27, no. 1, 93–110, 2004. Article (CrossRef Link).

[13] A. Shokripour, M. Othman, "Categorizing Researches about DLT in Ten Groups," *International Association of Computer Science and Information Technology*, pp. 45–49, 2009. Article (CrossRef Link).

[14] Y. Yang, H. Casanova, M. Drozdowski, M. Lawenda, A. Legrand, "On the Complexity of Multi-Round Divisible Load Scheduling," *Research Report RR*-6096, 2007. Article (CrossRef Link).

[15] A. R, J. Agarkhed, "Evaluation of Auto Scaling and Load Balancing Features in Cloud," *International Journal of Computer Applications*, vol.117 no.6, pp. 30-33, 2015. Article (CrossRef Link).

[16] B. Javadi, R. K. Thulasiram, R. Buyya, "Characterizing spot price dynamics in public cloud environments," *Future Generation Computer Systems*, vol 29, no 4, pp. 988-999, 2013. Article (CrossRef Link).

[17] J. Hwang, and J. Yoo, "FaST: Fine-grained and Scalable TCP for Cloud Data Center Networks," *KSII Transactions on Internet and Information Systems(TIIS)*, vol. 8, no. 3, pp.762-777, 2014. Article (CrossRef Link).

[18] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, "The globus striped ridftp framework and server," in *Proc. of the 2005 ACM/IEEE conference on* Supercomputing, SC '05, IEEE Computer Society, pp. 54–65, 2005. Article (CrossRef Link).

[19] L. Ramakrishnan, C. Guok, K. Jackson, E. Kissel, D. M. Swany, D. Agarwal, "On-demand overlay networks for large scientific data transfers," in *Proc. of 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 359 –367, 2010. Article (CrossRef Link).

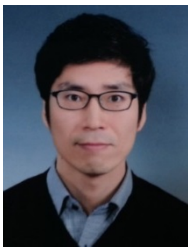
[20] D. Yin, E. Yildirim, S. Kulasekaran, B. Ross, T. Kosar, "A data throughput prediction and optimization service for widely distributed many task computing," *IEEE Transactions on Parallel and Distributed Systems*, vol.22, no.6, pp. 899 –909, 2011. Article (CrossRef Link).

[21] V. Garonne, A. Tsaregorodtsev, E. Caron, "A study of meta-scheduling architectures for high throughput computing: Pull versus push," *International Symposium on Parallel and Distributed Computing*, pp. 226–233, 2005. Article (CrossRef Link).

[22] J.T. Moscicki, "Diane - distributed analysis environment for grid-enabled simulation and analysis of physics data," in *Proc. of IEEE Nuclear Science Symposium Conference Record* vod. 3, pp. 1617–1620, 2003. Article (CrossRef Link).

[22] A. Tsaregorodtsev, V. Garonne, I. Stokes-Rees, "Dirac: A scalable lightweight architecture for high throughput computing," in *Proc. of GRID '04 Proceedings of the 5th IEEE/ACM InternationalWorkshop on Grid Computing*, pp. 19–25, 2004. Article (CrossRef Link).

[23] J. Diaz-Montes; M. Diaz-Granados; M. Zou; S. Tao; M. Parashar, "Supporting Data-intensive Workflows in Software-defined Federated Multi-Clouds," in *Proc. of IEEE Transactions on Cloud Computing*, vol.PP, no.99, pp.1-1 Sep, 2015. Article (CrossRef Link).
[24] Dandan Wang, Yang Yang and Zhenqiang Mi, "QoS-Based and Network-Aware Web Service Composition across Cloud Datacenters," *KSII Transactions on Internet and Information Systems(TIIS )*, vol. 9, no 3, pp.971-989, Mar. 2015. Article (CrossRef Link).
[25] Haoran Ji, Weidong Bao, Xiaomin Zhu and Wenhua Xiao, "Topology-based Workflow Scheduling in Commercial Clouds," *KSII Transactions on Internet and Information Systems(TIIS)*, vol.9, No. 11 pp.4311-4330, Nov. 2015. Article (CrossRef Link).
[26] B. Kim, C.-H. Youn, "A performance evaluation of the synchronized provisioning with adaptive buffer resilience scheme over grid networks," *IEEE Communications Letters*, vol. 16, no. 4, Apr, 2012. Article (CrossRef Link).
[27] C. Cassandras, Y. Wardi, B. Melamed, G. Sun, C. Panayiotou, "Perturbation analysis for online control and optimization of stochastic fluid models," *IEEE Transactions on Automatic Control*, vol. 47, no. 8, pp. 1234 – 1248, 2002. Article (CrossRef Link).
[28] Y. Zhao, B. Melamed, "Ipa derivatives for make-to-stock production inventory systems with backorders," *Methodology and Computing in Applied Probability* vol.8, pp. 191–222, 2006. Article (CrossRef Link).
[29] F. Howell, R. Mcnab, simjava: a discrete event simulation library for java, pp. 51–56, 1998. Article (CrossRef Link).
[30] J. A. Buzacott, J. G. Shanthikumar, Stochastic Models of Manufacturing Systems, *Prentice Hall*, 1993. Article (CrossRef Link).



**Byungsang Kim** received the B.Sc degree in Dongkuk University, Seoul, Korea, in 2002. He also received a Ph.D and M.Sc degree in Information and Communicatons Engineering KAIST, Daejeon, Korea from in 2013 and 2004 respectively. Since 2013, he had been worked on the Visual Display Division in Samsung Electronics. Where he had developed core technologies of voice search engine on the Smart TV. Now he is a senior researcher in Samsung Software R&D Center, Seoul. Also, he was a researcher in Korea e-Science research group in Super Computing Center in KISTI, Daejeon Korea, from 2006 to 2009 and a researcher in the Gri Middleware Center in KAIST from 2009 to 2010. His research fields are the large scale data processing platform, the core search engine, natural language processing, and cloud and grid computing technologies.