Applied Artificial Intelligence

An International Journal

ISSN: 0883-9514 (Print) 1087-6545 (Online) Journal homepage: https://www.tandfonline.com/loi/uaai20

Parallel and Distributed Powerset Generation Using Big Data Processing

Youssef M. Essa, Ahmed El-Mahalawy, Gamal Attiya & Ayman El-Sayed

To cite this article: Youssef M. Essa, Ahmed El-Mahalawy, Gamal Attiya & Ayman El-Sayed (2019) Parallel and Distributed Powerset Generation Using Big Data Processing, Applied Artificial Intelligence, 33:13, 1133-1156, DOI: 10.1080/08839514.2019.1665262

To link to this article: https://doi.org/10.1080/08839514.2019.1665262

Published online: 01 Oct 2019.

APPLIED ARTIFICIAL INTELLIGENCE
2019, VOL. 33, NO. 13, 1133–1156
https://doi.org/10.1080/08839514.2019.1665262

Parallel and Distributed Powerset Generation Using Big Data Processing

Youssef M. Essa[a], Ahmed El-Mahalawy[b], Gamal Attiya[b], and Ayman El-Sayed[b]

[a]Senior BigData Engineer, Idealo, Berlin, Germany; [b]Faculty of Electronic Engineering, Computer Science and Engineering Department

ABSTRACT
Data mining algorithms are more important today as it allows stakeholders to get a 360-degree view of their customers. Recently, powerset has become the basic core for many algorithms and techniques in different data mining domains as it provides optimal solutions for many problems in data mining. Nevertheless, it is challenging to be used in several instances because the complexity of powerset grows exponentially with the number of sets. Constructing powerset from huge datasets on a single machine causes an out-of-memory exception. So, from a business perspective in mega data projects, the enterprise companies need to invest a lot of money to build high-performance system infrastructure of powerset. Also, enterprise companies have to invest more money to build a standby system to keep the system alive if the high-performance machines break down. Furthermore, the existing powerset techniques are designed for structured data and not useful in intensive processing using in-memory unstructured data store. Thus, this paper tackles most problems that hinder deploying powerset algorithm toward Big Data and presents a series of pruning techniques that can greatly improve construction efficiency of powerset generation. The approach allows enterprise companies to explore huge data volumes and gain business insights into near-real-time and save the cost of infrastructure.

Introduction

Knowledge discovery from data refers to a set of activities designed to extract new knowledge from complex datasets (Wu et al. 2014). In recent years, large quantities of data have become increasingly available through many data sources such as Internet of Things, databases, and social media. In fact, enterprise firms are investing hundreds of millions of dollars every year in knowledge extraction in order to support their customers' decisions in the current fast-changing business environments (Kiron et al. 2012). However, because of the difficulty of processing huge data, decision-makers become frustrated to get answers to several questions and insights into near-real-time. This issue drew the attention of the enterprise companies toward the emerging and interesting use of parallel or distributed systems like big data processing platforms or power systems (Essa, Attiya, and

CONTACT Youssef M. Essa yosufessa@gmail.com Berlin, Germany
Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/uaai.

El-Sayed 2014; Wang et al. 2016). Big data is a term for datasets that are so large or complex that traditional data processing applications are inadequate to deal with them. The 4 V's (volume, variety, velocity, and veracity) imply that the data is hard to handle with an acceptable performance and low cost by using a single machine and traditional algorithms (Essa, Attiya, and El-Sayad 2013). In big data, a large application is divided into smaller units, called tasks, and those tasks have to be assigned to appropriate processing elements in the distributed system, to be concurrently processed. This behavior can dramatically improve performance because each task will be assigned and executed on the best-suited machine (Salah, Akbarinia, and Masseglia 2015).

Powerset algorithm is one of the algorithms used to extract features in data mining (Esfandiari and Babavalian et al. 2014) and generate and select unique features (Spolaôr and et al. 2013). However, there are many challenges to use powerset algorithms with huge datasets. The challenges raised by the dataset volumes, velocity data, and variety of data (Seol and Jeong et al. 2013). The data volumes cause an out-of-memory exception when running powerset algorithm in commodity machine because the maximum size of memory (virtual memory space) is limited to 2–60-GB commodity machines (Vavliakis et al. 2014). In addition, scalability is another problem when needing to expand hardware infrastructure for a specific machine while generating powerset. The recent studies of powerset are using a single machine for deploying small dataset or using an existing or a custom simulator for the big dataset to save cost instead of using a high-performance machine. Membrane simulator (Ciobanu and Paraschiv 2002) and distributed software simulators (Ciobanu and Wenyuan 2003) are examples of simulators which are used to execute big dataset. So, this paper tackles most problems that hinder deploying powerset algorithm toward big Data distributed computing system instead of executing powerset on a single machine.

The algorithm mentioned in this paper breaks all dependencies between partial powerset to execute all of them concurrently and collected all results also in parallel without any type of dependency. This approach will save industry cost when deploying big dataset on the distributed cluster instead of a single machine. Also, this approach provides replications of partial dataset over the machines in the cluster, and this feature is not valid when deploying on a single machine because a single machine failure is considered as a breakpoint.

The rest of this paper is organized as follows. Related work presents the most recent related work to powerset execution algorithms. Problem Statement presents the problem statement for current powerset generation algorithms. A Novel Theory of Powerset Generation section introduces novel theories for powerset generation. then, the next section presents proposed powerset algorithms. Performance Evaluation section presents a comparative study of different approaches. After that, Discussion section discusses experimental discussion and several key points. Finally, the paper is concluded in Conclusion Section.

Related Work

Several models were developed for powerset generation. The readers may refer to Amailef and Jie (2013), Bahnasy, Naguib, and Aref (2014), Rashedi, Nezamabadi, and Saryazdi (2013), and Zhou et al. (2017) for more details on generating powerset in sequential model. In Bahnasy, Naguib, and Aref (2014), an approach for indexing large case bases is discussed using powerset tree. First, unique reductions for each case over all cases are obtained and then another case base of those unique features with reference to original cases is built. In Rashedi, Nezamabadi, and Saryazdi (2013), a feature selection and classification approach is presented. A rough set algorithm based on powerset tree was applied for feature selection from 20 features based on shape, color, and texture. They have been extracted in order to obtain a feature vector for each object to identify and classify the tumor using 2D brain Magnetic Resonance Imaging. In Amailef and Jie (2013), the authors discussed a novel case base indexing model to improve the performance of indexing and retrieving in the data warehousing. A fully customized solution has been designed and built to find the unique combinations to each case and these unique combinations are used to build the case base index. From cost and performance perspective, the sequential finding powerset is the best approach to generate powerset in small scale. However, the data projects today have a very large or huge amount of data in datasets. So, the only way to enhance the sequential process is by replacing old machine with new higher specs regarding the size of data. Definitely, this approach is not flexible enough in the mega project, and enterprise companies need to invest a big amount of money to process a big amount of data. Furthermore, from performance perspective, this approach could increase the complexity to achieve the business goals to process big dataset in nearest or in real time. Gil et al. (2008) introduced an active rules application algorithm based on powerset (P) of active Rules (R). The powerset of R is denoted by P(R). The elimination of rules is shown in Algorithm 1. This algorithm depends on the number of rules of the membrane. So, it provides powerset in parallel and execution time is delimited. The Transition P systems perform a computation between two consecutive configurations through transition or evolution step. Then, each step is made via two phases. The first phase is performing evolution rules. The second phase is communication between membranes.

Algorithm 1: Forming P(R) elements.

```
Ensure: User defined code will be executed for each element of E ∈ P(R).
    1. Member of P(R) initialization REPEAT
    2. Multiset proposition
    3. Selection of P(R) element
    4. Checking rules halt UNTIL End
    5. end procedure
```

After F.J.Gil working, the (Zhou et al. 2017), used recursion algorithm based on masking approaches to obtain more accurate prediction results. The authors developed the new approach called PHBRB for projection covariance matrix adaption based on power set to generate the constrained elemental cardinality powerset P(S) from the given set (S), as shown in Algorithm 2, forming Pj(S).

Algorithm 2: Forming Pj(S).

Require: $j \geq 0$; an array S s.t. length(S) $\leq j$; a temporary array E of size j.

Ensure: User defined code will be executed for each element of $E \in Pj(S)$.

```
1. Powersetj (S, j)                          ◁ Start the algorithm
2.  procedure powersetj (U, s)
3.  if s > 0 then                              ◁ Recur deeper
4.        l ← length(U)
5.        for i ← 1, l − s + 1 do
6.             E(s) ← U(i)
7.              powersetj (U(1 + i: l), s − 1)    ◁ Recursion
8.        end for
9.   else
10.        User defined handling of E representing E ∈ Pj(S).
11.  end if
12.  end procedure
```

This approach generates powerset using the parallel idea that is based on PHBRB's masking. The algorithm finds Pj(S) which is the largest subset of P(S) containing all its elements of cardinality equal to j. In this algorithm, the masking approaches are used to generate the powerset P(S) of a set S. If the cardinality of the set S is n, then n digits represent each subset of S. Each digit position in the binary number is assigned to a specific element of S. If the position contains 1, this means the existence of the element in the corresponding subset; otherwise, the element is not included in the subset.

In Sahakyan (2014), an approach based on lexicographical order is developed to generate the subset of the powerset masked by binary numbers starting from zero number 000 … 0 and gradually increasing in the number value until reaching the number 111111 … 1. Using this method for generating the powerset has some drawbacks. For instance, the element corresponding to the most significant bit will not appear before half of all of the subsets being generated which is not suitable when the value of n is large.

The algorithms in Gil et al. (2008), Sahakyan (2014), and Zhang, Tianrui, and Pan (2012) used a hybrid model between sequential and parallel models to generate powerset using two phases. The first phase split dataset into multiple rules running concurrently and the results of those rules are dependent on each other in the second phase; repeat this operation until all rules are applied. Indeed, this approach enhanced performance from the first fully sequential execution. However, hybrid model didn't solve breakpoint issue and still depends on generating powerset on a single machine. Also, this approach highly cost in the big dataset.

This paper presents a novel theory for powerset generation based on big data processing concepts to improve performance, reliability, and scalability of generating powerset over a distributed computing system. The main idea is based on increasing memory and powerful parallel and distributed processing to crunch large volumes of data extremely quickly, where many machines are used to solve a problem. The approach allows enterprise companies to explore huge data volumes and gain business insights into near-real-time and save the cost to build infrastructure especially for enterprise companies.

Problem Statement

Today, companies, institutions, health-care systems, etc. use piles of data. This huge data is further used for creating reports in order to ensure continuity regarding the services that they have to offer. The challenge is how to manipulate or analyze an impressive huge volume of data to find commercially valuable insights using feature extraction algorithms. Powerset algorithm is one of the algorithms that is used to solve feature selection problem in data mining. However, the main challenge for using powerset algorithms in big data scale is the complexity of processing (Zhang, Tianrui, and Pan 2012). Algorithm 2 presents the problem of complexity of powerset processing and the complexity of the power set algorithm is O(2^n) and needs more hardware resources to increase performance. To prove that, we assumed i, j, and w for each iteration of the outer loop j, given set size with length(s) = n, the following generalized equations for i, j, and w:

$$\mathrm{j} = \mathrm{n}$$

$$\mathrm{i} = \frac{n(n-1)}{2}$$

$$w = \sum_{j=0}^{n}\left(\sum_{i=0}^{j-2} 2^j - 1\right)$$

So, from the above equations, the subsets generated from Set of n elements are equal to

$$(j+i+w)=\sum_{j=0}^{n}\left(\sum_{i=0}^{j-2}2^{j}-1\right)+\frac{n(n-1)}{2}+n=2^{n}-1$$

Thus, it explains the problem of the complexity O(2^n) of the power set algorithm. The challanges to process huge data using powerset algorithm raised by the data volumes, complexity, and variety. The enterprise companies are investing a lot in monetary terms to set up high-performance systems with standby system to address the complexity of constructing machine learning. Therefore, powerset processing in big data scale needs new execution strategies, and this article introduces a new theorem to accelerate the process of powerset generation. The main idea is breaking dependencies between set elements and construct powerset using big Data processing platforms such as Hadoop or Spark. Big data processing platforms drive dramatic increases in performance by solving latency problems as it processes big data in real time (Hu et al. 2014; Jiang et al. 2015; Khalifa et al. 2016; Philip and Zhang 2014).

A Novel Theory of Powerset Generation

This section provides a novel theory for the powerset generation proposed in this paper and the proof of this theory as well using a binary representation model (Bova, Kureichik, and Lezhebokov 2014). This novel theory proposes a new method to reduce the complexity to O(n). This is achieved by partitioning the set S into smaller subsets to be able to process these subsets over parallel or distributed computation models instead of sequential models. This processing approach utilizes parallel architecture and would save the cost of powerset implementations by running all subset among commodity machines instead of high-performance systems. Additionally, this approach would provide better performance using parallel execution. In the next section, we prove that the dataset S can be partitioned into subsets and can be made to run them all in parallel with a complexity O(n) and then collect the sub-powerset of each one and use it to construct the powerset P(S).

Theorem:

Given A be a set of n elements such that S(A) $=\{a_1a_2a_3\ \ldots\ a_n\}$. To find powerset of P(A), the main step is to find all possible combinations of elements concurrently at the same time regarding distinct elements by splitting combination process into multiple of the parallel process as follows:

$$C(a^1)= \text{Combination elements of } S(a^1) \text{ based on } a_1.$$
$$C(a^2)= \text{Combination elements of } S(a^2) \text{ based on } a_2.$$
$$C(a^3)= \text{Combination elements of } S(a^3) \text{ based on } a_3.$$
$$\ldots$$
$$C(a^n)= \text{Combination elements of } S(a^n) \text{ based on } a_n.$$

Then, the powerset of A is given by

$$P(A) = C(A^1) \cup C(A^2) \cup C(A^3) \ldots \cup C(A^n) \cup \emptyset.$$

For example, given a set $S(A) = \{a, b, c\}$, then the combinations of elements regarding to each element are as follows:

$$C(a) = \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}.$$
$$C(b) = \{\{b\}, \{b, c\}\}$$
$$C(c) = \{c\}$$

The powerset of P(A) has 2^3 elements as follows:

$$P(A) = \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}, \{b\}, \{b, c\}, \{c\}, \emptyset\}$$

Proof of theorem:

The proof of theorem is done by using the binary representation of the subsets (Bova, Kureichik, and Lezhebokov 2014). Let A be partitioned into two subsets B and D, where $B = \{c_1, c_2, c_3, \ldots, c_{m1}\}$ and $D = \{c_2, c_3, \ldots, c_{m1}\}$ with $|B| = m_1$ and $|D| = m_1 - 1$. Any subset $B_i \in P(B) : 1 \le i \le 2^{m_1}$ can be represented by the binary number $\overline{\overline{c_1c_2c_3 \ldots c_{m_1}}}$ such that $\overline{\overline{b_i}} = \begin{cases} 1: & c_i \in B_i \\ 0: & c_i \notin B_i \end{cases}$ and $B_i = S(\overline{\overline{c_1c_2c_3 \ldots c_{m_1}}})$.

If the lexicographical order of binary numbers of length $m_1$is used starting with$[\![000 \ldots 0]\!]_{m_1}$and ending with $[\![111 \ldots 1]\!]_{m_1}$, all subsets of P(B) will be generated. The same for the powersets P(D), all its subsets $D_i \in P(D) : 2 \le i \le 2^{m_1}$can be represented using the lexicographical order of binary numbers $\overline{\overline{c_2c_3 \ldots c_{m_1}}}$of length m_2 from $[\![000 \ldots 1]\!]_{m_1}$to $[\![111 \ldots 1]\!]_{m_1}$. From the definition of $\langle\cup\rangle$, obtain thecombination C_n based on n of $B\langle\cup\rangle D = \{C_i(B_i) \cup C_j(D_j) : \forall_{i,j} : 1 \le i \le m_1 \text{and} 2 \le j \le m_1\}$.

Since B and D form a partition of A and since $\overline{\overline{c_1c_2c_3 \ldots c_{m_1}}}$ and $\overline{\overline{c_2c_3 \ldots c_{m_2}}}$ are the binary representation of any subset from B and D, respectively, which means that $S(\overline{\overline{c_1c_2c_3 \ldots c_{m_1}}}) \in P(A)$ and $S(\overline{\overline{c_2c_3 \ldots c_{m_2}}}) \in P(C)$, moreover the binary number of length n defined by the Combination of $C(\overline{\overline{c_1c_2c_3 \ldots c_{m_1}}})$ $\overline{\overline{C(c_2c_3 \ldots c_{m_2})}}$ represents a subset of the powerset P(A) such that

$S\left(\overline{\overline{c_1c_2c_3\ \ldots\ c_{m_2}}}\right) = C\left(\overline{\overline{c_1c_2c_3\ \ldots\ c_{m_1}}}\right) \mathrm{U}\, C\left(\overline{\overline{c_2c_3\ \ldots\ c_{m_1}}}\right) \mathrm{U}\, C\left(\overline{\overline{c_3\ \ldots\ c_{m_1}}}\right) \ldots$ $\mathrm{U}\, C\left(\overline{\overline{c_{m_1}}}\right) \in P(A)$.

Table 1 proves that $P(A) = C\left(A^1\right)\langle\cup\rangle C\left(A^2\right)$.

Now, when assuming $k = k_0$, if A is partitioned into k_0 subsets $A^1,\ A^2,\ A^3,\ \ldots\ ,\ A^{k_0}$, the powerset $P(A)$ of A is given by

$$P(A) = P\left(A^1 \cup A^2 \cup\ \ldots\ \cup A^{k_0}\right) = P\left(A^1\right)\langle\cup\rangle P\left(A^2\right)\langle\cup\rangle\ \ldots\ \langle\cup\rangle P\left(A^{k_0}\right)$$

So, it is essential to prove that if A is partitioned into $k_0 + 1$ subsets $A^1,\ A^2,\ A^{k3},\ \ldots\ ,\ A^{k_0}, A^{k_0+1}$, then the powerset $P(A)$ of A is given by

$$\begin{aligned} P(A) &= P\left(A^1 \cup A^2 \cup\ \ldots\ \cup A^{k_0} \cup A^{k_0+1}\right) \\ &= P\left(A^1\right)\langle\cup\rangle P\left(A^2\right)\langle\cup\rangle\ \ldots\ \langle\cup\rangle P\left(A^{k_0}\right)\langle\cup\rangle P\left(A^{k_0+1}\right) \end{aligned} \tag{1}$$

Since A is partitioned as $A^1,\ A^2,\ A^3,\ \ldots\ ,\ A^{k_0}, \mathrm{and} A^{k_0+1}$, then,

$$A^i \neq \emptyset,\ \forall 1 \leq i \leq k_0 + 1$$

$$A = \mathrm{U}_{i=1}^{k_0+1} A^i$$

$$A^i \cap A^j = \emptyset,\ \forall i \neq j,\ 1 \leq i \leq k_0 + 1,\ 1 \leq j \leq k_0 + 1$$

From which we deduce that if $\widetilde{A} = \mathrm{U}_{i=1}^{k_0} A^i$ then,.

$$A^i \neq \emptyset,\ \forall 1 \leq i \leq k_0$$

$$\tilde{A} = \mathrm{U}_{i=1}^{k_0} A^i$$

$$A^i \cap A^j = \emptyset,\ \forall i \neq j,\ 1 \leq i \leq k_0, 1 \leq j \leq k_0.$$

Table 1. Combination of two subsets.

Element representations of subset $A_i \in P(A)$	Element representations of subset $B_i \in P(B)$	Element representations of subset $C_i \in P(D)$
$S\left(\overline{\overline{c_1c_2c_3\ldots c_{m_2}}}\right)$	$S\left(\overline{\overline{b_1b_2b_3\ldots b_{m_1}}}\right)$	$S\left(\overline{\overline{c_2c_3\ldots c_{m_1}}}\right)$
$\{\emptyset\}$	$\{\emptyset\}$	$\{\emptyset\}$
$\{c_{m_2}\}$	$\{\emptyset\}$	$\{c_{m_2}\}$
$\{c_{m_{2-1}}\}$	$\{\emptyset\}$	$\{c_{m_{2-1}}\}$
$\{c_{m_{2-1}}, c_{m_2}\}$	$\{\emptyset\}$	$\{c_{m_{2-1}}, c_{m_2}\}$
$\{c_{m_{2-2}}, c_{m_{2-1}}, c_{m_2}\}$	$\{\emptyset\}$	$\{c_{m_{2-2}}, c_{m_{2-1}}, c_{m_2}\}$
$\ldots$	$\ldots$	$\ldots$
$\{c_1, c_2, c_3, \ldots, c_{m_{2-2}}, c_{m_{2-1}}, c_{m_2}\}$	$\{\emptyset\}$	$\{c_1, c_2, c_3, \ldots, c_{m_2}\}$
$\{b_{m_1}\}$	$\{b_{m_1}\}$	$\{\emptyset\}$
$\{b_{m_1}, c_{m_2}\}$	$\{b_{m_1}\}$	$\{c_{m_2}\}$
$\ldots$	$\ldots$	$\ldots$
$\{b_1, b_2, b_3, \ldots, b_{m_1}, c_1, c_2, c_3, \ldots c_{m_{2-2}}\}$	$\{b_1, b_2, b_3, \ldots, b_{m_1}\}$	$\{c_1, c_2, c_3, \ldots c_{m_{2-2}}\}$
$\{b_1, b_2, b_3, \ldots, b_{m_1}, c_1, c_2, c_3, \ldots c_{m_{2-2}}, c_{m_2}\}$	$\{b_1, b_2, b_3, \ldots, b_{m_1}\}$	$\{c_1, c_2, c_3, \ldots c_{m_{2-2}}, c_{m_2}\}$
$\{b_1, b_2, b_3, \ldots, b_{m_1}, c_1, c_2, c_3, \ldots, c_{m_{2-1}}\}$	$\{b_1, b_2, b_3, \ldots, b_{m_1}\}$	$\{c_1, c_2, c_3, \ldots, c_{m_{2-1}}\}$
$\{b_1, b_2, b_3, \ldots, b_{m_1}, c_1, c_2, c_3, \ldots, c_{m_2}\}$	$\{b_1, b_2, b_3, \ldots, b_{m_1}\}$	$\{c_1, c_2, c_3, \ldots, c_{m_2}\}$

Therefore, $\tilde{A}$ is partitioned by the k_0 subsets $A^1,\ A^2,\ A^3,\ \ldots\ ,\ A^{k_0}$. Using the hypothesis, we conclude that.

$$P(\tilde{A}) = P\left(A^1 \cup A^2 \cup \ \ldots\ \cup A^{k_0}\right) = P\left(A^1\right)\langle\cup\rangle P\left(A^2\right)\langle\cup\rangle\ \ldots\ \langle\cup\rangle P\left(A^{k_0}\right).$$

Now, since $\tilde{A} = A^1 \cup A^2 \cup \ \ldots\ \cup A^{k_0}$ and A^{k_0+1}satisfy the partitioning conditions with respect to the set A:

$$\tilde{A} \neq \emptyset \text{ and } A^{k_0+1} \neq \emptyset$$
$$A = \tilde{A} \cup A^{k_0+1}$$
$$\tilde{A} \cap A^{k_0+1} = \emptyset$$

Then using Equation (1), it is directly concluded that

$$P(A) = P\left(\tilde{A} \cup A^{k_0+1}\right) = P\left(\tilde{A}\right)\langle\cup\rangle P\left(\ A^{k_0+1}\right)$$

But, $\tilde{A} = A^1 \cup\ A^2\ \cup\ \ \ldots\ \ \cup A^{k_0}$, then

$$P(A) = P\left(A^1 \cup^{A^2} \cup\ \ldots\ \ \cup^{A^{k_0}}\right)\ \langle\cup\rangle P\left(A^{k_0+1}\right)$$
$$= P\left(A^1\right)\langle\cup\rangle P\left(A^2\right)\langle\cup\rangle \ldots\ \ \langle\cup\rangle\ P\left(A^{k_0}\right).\langle\cup\rangle\ P\left(A^{k_0+1}\right)$$

Hence, the ability of this theorem is to produce p(S) from a sub-powerset using content Union operation. Additionally, this approach of execution reduces the complexity to O(n) and provides better performance than sequential execution approach.

Proposed Powerset Algorithms

In this section, the theorem presented in Section III is used to generate the powerset of a given set S that contains n elements considering three different models of computation.

The first model is parallel computation based on high-performance computing (Pronk et al. 2015). Algorithm 3 shows how to use theorem for generating the powerset by partitioning the given Set S into S[m] subsets. The number of subsets represented by m. So, after computing the powerset for each subset, the set contents Union operation $\langle\cup\rangle$ used to generate the final powerset of the set S.

Algorithm 3: Forming powersets using parallel processing.

```
Require: input set (S), a temporary array E of size j.
Ensure: User defined code will be executed for each element of $E \in Pj(S)$
to find P(S).
    1. Partitioning S into subsets S[m]
    2. for i = 0 to size of m do
    3.    if $(0 < |S_m| \le m) S[i] = d$ //d is number of elements in subset
    4.    END if
    5. END for
    6. parallel loop j = 0 to d do
    7. load balancer to utilize h/w resources between subsets
    8. $E[S^i]$ = all possible combination of basic element for $S[m^j]$.
    9. END for
    10. Combine results from all tasks to find powerset using
        Union operation
```

Algorithm 4: Forming powersets using MapReduce processing.

```
Require: input set (S), a temporary array E of size j on each mapper.
Ensure: User defined code will be executed for each element of E
$\in Pj(S)$ to find P(S).
    1.  Partitioning S into subsets S[m]
    2.  for i = 0 to size of m do
    3.     if $(0 < |S_m| \le m) S[i] = d$ //d is number of elements in subset
    4.     END if
    5.  END for
    6.  Mapping S[m] to Mapper Machines
    7.  Mapper (Input S[i], output $P(S_i)$)
    8.  E[S] = all possible combination of basic element for S[m]
    9.  END mapper
    10. Reducers receive sub-powerset
    11. Reducer (Input $E[S_j]$, output P(S))
    12. Combine results from all tasks to find powerset using
        Union operation
    13. END Reducer
```

The second model is distributed computation model based on big data processing platform using Hadoop. Algorithm 4 generates the powerset by portioning a given set into subsets and mapping subsets among different mappers in the cluster. The mappers' machines compute the powerset for each subset. Then, the reducer combines sub-powersets using the set contents Union operation $\langle\cup\rangle$ to generate the final powerset of the set S.

The third model is a distributed computation model based on the big data stream processing platform. Spark is the most popular platforms used in stream processing (Cristian et al. 2016). These platforms allow running stream processing code across distributed machines. Stream processing model is designed to generate powerset on real-time streaming data, as shown in Algorithm 5. The main idea is that portioning powerset to subsets using powerset theorem and sending each portion as an event to stream processing platform. Each subset uses all machines in the cluster to compute the value of the subset. At the same time, the value of subset is used to update final powerset of the set S using the set contents Union operation $\langle\cup\rangle$.

Algorithm 5: Forming powersets based on streaming processing.

Require: input set (S), a temporary array E of size i on each machine regarding to event.

Ensure: User defined code will be executed for each element of E ∈ Pj(S) to find P(S).

```
1.   Partitioning S into subsets S[m] and send each subset after reading
2.   for i = 0 to size of m do
3.     if $(0 < |S_m| \le m) S[i] = d$ //d is number of elements in subset
4.   END if
5. END for
6. Start stream processing program to process events subset $E[S^i]$
7. Distributed subset into distributed cluster
8. Each machine generates list of possible elements
9. Combine results from all executed tasks to find sub-powerset using Union operation
10. Update final powerset P(s)using value of current events
11. END Stream Processing.
```

Performance Evaluation

This section presents the results of the practical experiments that are done to evaluate the proposed algorithms for parallel and distributed computations of powerset generation. The implementation using different sizes of dataset and each element of the set is a text value between 100 and 256 characters and set size in all figures represents the number of elements in each dataset.

Implementation Environment

The implementation environment consists of hardware and software components with specifications shown in Table 2.

(1) Hardware Components:

High-performance system and Hadoop cluster are used in the practical implementation. The high-performance computing (HPC) of IBM power system E850 is used to implement the parallel algorithm. On the other hand, the Hadoop cluster runs over homogenous machines that have different hardware and software specifications. Hadoop cluster consists of 50 machines connected with each other through a Local Area Network. The Hadoop cluster is a homogenous cluster, this means all machines have equal hardware specifications and workload distributed equally among all machines. Also, the all process executed in-memory in spark as well as HPC.

(2) Software Components:

Message Passing Interface (MacArthur et al. 2017) is used to develop F.J.Gil parallel computation algorithm of powerset, as shown in Figure 1. Furthermore, the Hadoop platform used to develop distributed computation is shown in Figure 2. In addition, Hadoop is a basic platform to run Spark streaming to develop streaming distributed computation as shown in Figure 3.

The powerset application is being applied to each platform. It is a simple program that creates a dataset of size given by the user. Then, it divides the set into subsets that are processed on parallel and distributed environments.

In parallel processing, the high-performance machine first creates a number of tasks for each subset equivalent to the number of basic elements and runs all tasks concurrently. The next step is to combine all results executed from different tasks

Table 2. Software and hardware equipments.

	IBM power system	Hadoop cluster (homogenous cluster)
Model	IBM E850	Hadoop and spark
Number of machines	1	30
CPU	50 cores, 3.02 GHz	50 cores, 3.02 GHz
FPGA components	12	N/A
RAM	320 GB	300 GB
Operating system	Linux	Linux.
Network transfer rate	192 GB/s	s10 GB/

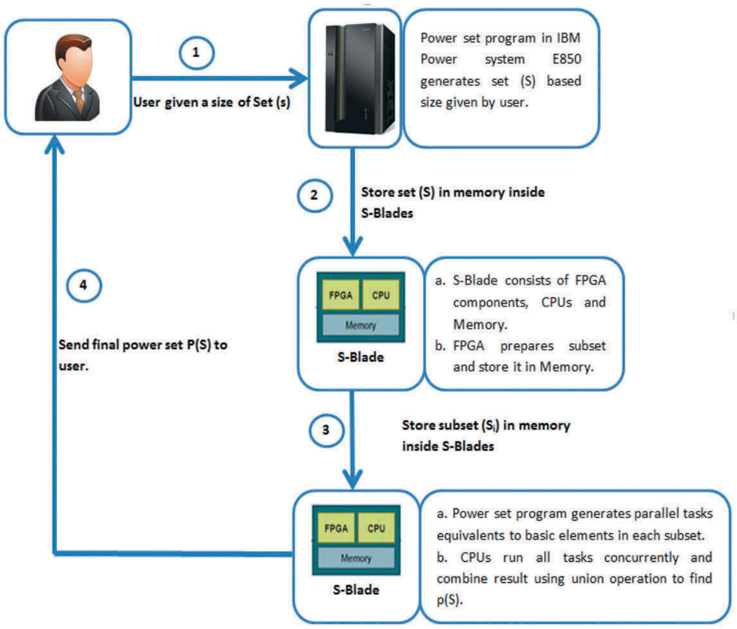

1
User given a size of Set (s)
Power set program in IBM Power system E850 generates set (S) based size given by user.
2
Store set (S) in memory inside S-Blades
FPGA
CPU
Memory
S-Blade
a. S-Blade consists of FPGA components, CPUs and Memory.
b. FPGA prepares subset and store it in Memory.
3
Store subset (S_i) in memory inside S-Blades
a. Power set program generates parallel tasks equivalents to basic elements in each subset.
b. CPUs run all tasks concurrently and combine result using union operation to find p(S).
4
Send final power set P(S) to user.

Figure 1. Powerset generation using high-performance computing.

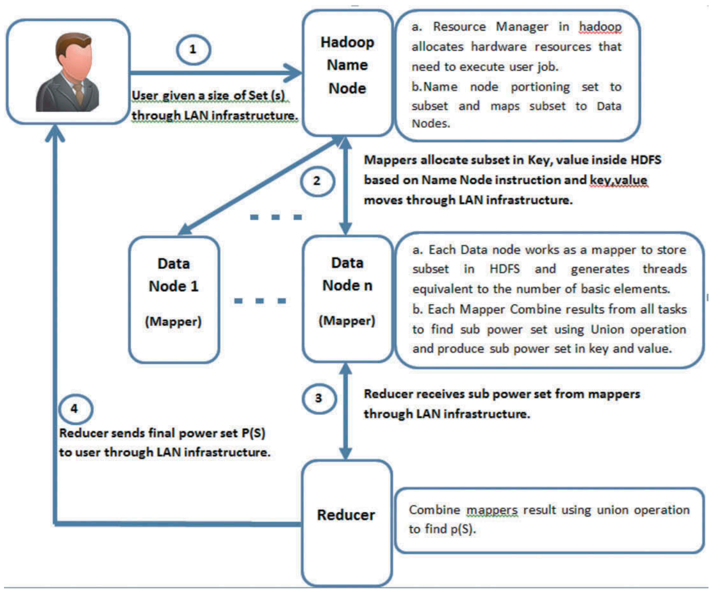

1
User given a size of Set (s) through LAN infrastructure.
Hadoop Name Node
a. Resource Manager in hadoop allocates hardware resources that need to execute user job.
b.Name node portioning set to subset and maps subset to Data Nodes.
2
Mappers allocate subset in Key, value inside HDFS based on Name Node instruction and key,value moves through LAN infrastructure.
Data Node 1 (Mapper)
Data Node n (Mapper)
a. Each Data node works as a mapper to store subset in HDFS and generates threads equivalent to the number of basic elements.
b. Each Mapper Combine results from all tasks to find sub power set using Union operation and produce sub power set in key and value.
3
Reducer receives sub power set from mappers through LAN infrastructure.
4
Reducer sends final power set P(S) to user through LAN infrastructure.
Reducer
Combine mappers result using union operation to find p(S).

Figure 2. Powerset generation using MapReduce.

using Union operation. The final step is using Union operation to combine sub-powerset to find final P(S).

In a distributed system approach, MapReduce is used to run across a distributed system and Spark is used to run complex event processing across Hadoop distributed system. MapReduce uses the Map function to distribute

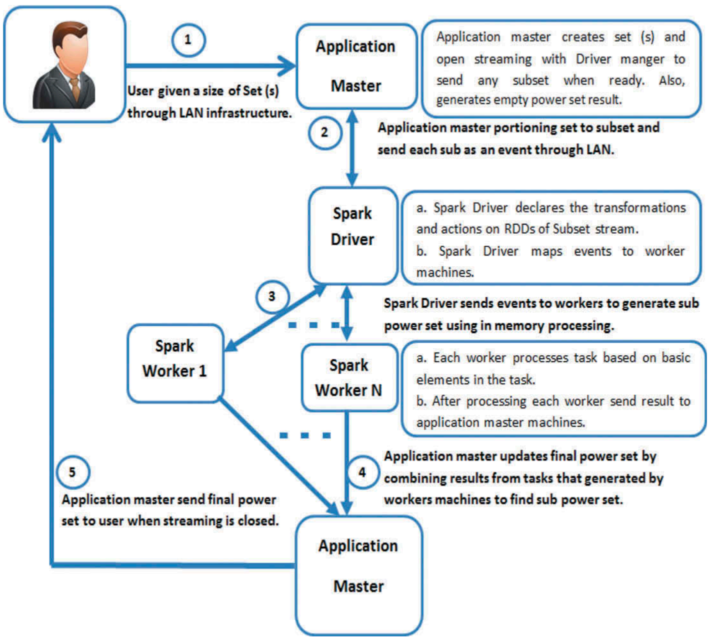

1
User given a size of Set (s) through LAN infrastructure.
Application Master
Application master creates set (s) and open streaming with Driver manger to send any subset when ready. Also, generates empty power set result.
2
Application master portioning set to subset and send each sub as an event through LAN.
Spark Driver
a. Spark Driver declares the transformations and actions on RDDs of Subset stream.
b. Spark Driver maps events to worker machines.
3
Spark Driver sends events to workers to generate sub power set using in memory processing.
Spark Worker 1
Spark Worker N
a. Each worker processes task based on basic elements in the task.
b. After processing each worker send result to application master machines.
4
Application master updates final power set by combining results from tasks that generated by workers machines to find sub power set.
5
Application master send final power set to user when streaming is closed.
Application Master

Figure 3. Powerset generation using spark streaming.

portioning subset to different mapper's machines in Hadoop cluster. Mapper's machine reads the input subset as <key,value> pairs and emits key-value pair, while the output from each mapper is key-value pairs where one subset is the key and the power subset is the value. Then, the Reducer's machine reads the outputs generated by the different mappers as <subset, power_subset> pairs and emits key-value pairs. The final output from Reducer is key-value pairs, where set (S) is the key and the final powerset P(S) is the value.

On the contrary, the Spark platform distributes each subset as an event among distributed clusters over the Hadoop distributed file system. Apache Spark receives each subset when created in the master machine. Each subset represents an event in Apache Spark for processing in Hadoop cluster. The main step is using Theorem in Section III to distribute tasks of subset among cluster to find the power of subset. In addition, during the processing of all events, the final powerset is updated by using the power subset of each subset as explain in our theorem.

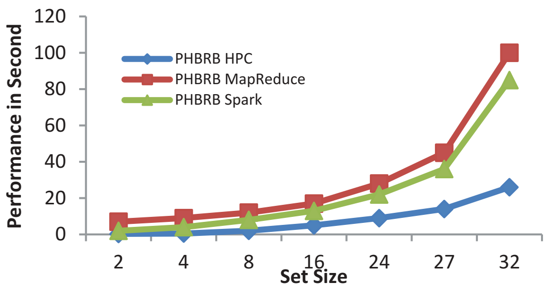
Real Implementation Results

In this section, performance evaluation is carried out by measuring the elapsed time for powerset generation by PHBRB (Zhou et al. 2017) and proposed approach considering different sizes of datasets. At a dataset size, the elapsed time is measured after applying the algorithm to generate powerset on a big data processing platform. Three different platforms are used in this evaluation.

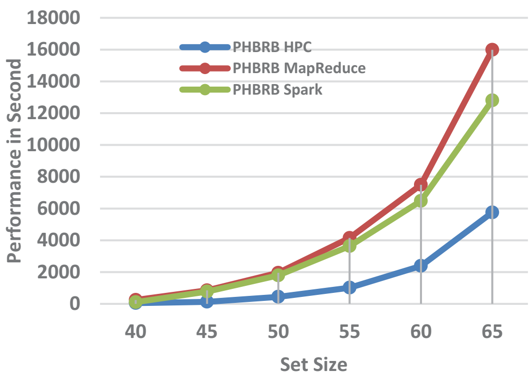
The platforms include distributed computing using Hadoop cluster and parallel computing using the high-performance machine IBM power system E850.

Figures 4 and 5 show the elapsed time for powerset generation by the PHBRB and the proposed approach, respectively, at different dataset sizes. The PHBRB algorithm supports parallel processing for subsets. It provides parallel processing for each subset and moves sequentially between them. On another hand, the proposed algorithm is responsible for processing datasets with a parallel and distributed computing on a cluster. Briefly, big data processing platform based on MapReduce contains two important tasks: Map and Reduce. The Map takes a subset of data and converts it into tuples (key-value pairs) and this operation is repeated sequentially based on a number of subsets. Then, the reduce stage takes the output from a map as an input and combines the data tuples into the smaller set of tuples.

As shown in Figures 4 and 5, the elapsed time by both the algorithms when using the HPC is less than that of using distributed computing based on MapReduce or Spark Hadoop cluster. This is because both the MapReduce and Spark spend additional time in the initialization process of cluster and mapping different tasks onto different machines on the cluster.

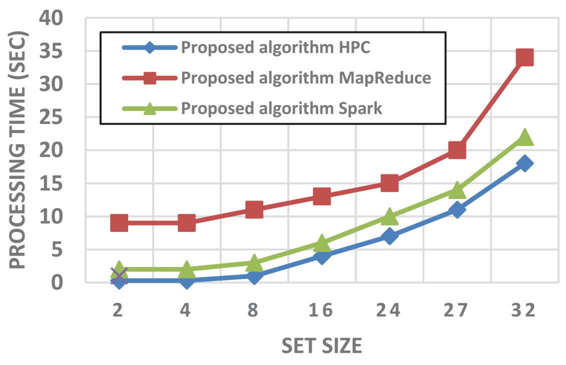


(a) Small size dataset.

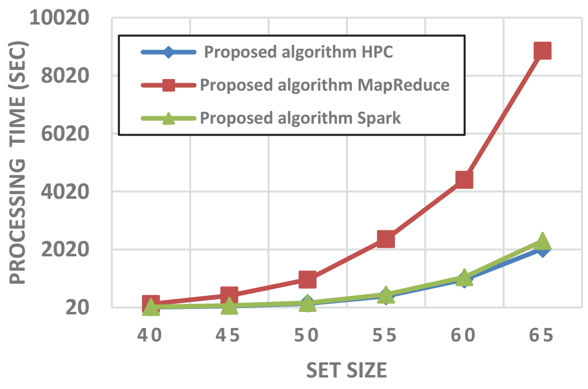


(b) Big size dataset.

Figure 4. PHBRB performance for powerset generation. (a) Small size dataset. (b) Big size dataset.



(a) Small size dataset.



(b) big size dataset.

Figure 5. Proposed algorithm for powerset generation (a) Small size dataset. (b) Big size dataset.

Furthermore, more time is required to map tasks onto machines through the network infrastructure. By contrast, the HPC acts as a single machine and does not need to do initialization process but can run any task directly on the available resources. In addition, all tasks run inside a single machine without needing to move through the network. However, the cost of Hadoop cluster is cheaper than HPC as it uses commodity hardware. In addition, Hadoop cluster is more flexible to expand hardware resources easily without the need for hard administration efforts.

Figure 6 shows the elapsed time for powerset generation by the PHBRB and the proposed approach, respectively, at different dataset sizes. From Figure 6, at small dataset size, less than 24 elements, there is no difference in processing time between PHBRB and proposed algorithm. However, at large dataset size, more than 24 elements, it is clear that the processing time of the proposed algorithm is less than that elapsed by the PHBRB algorithm. This is because the PHBRB algorithm cannot provide parallelism in the job execution. For this reason, the sequential moving between subsets spends more time when applying PHBRB in distributed platform with Spark or MapReduce because PHBRB

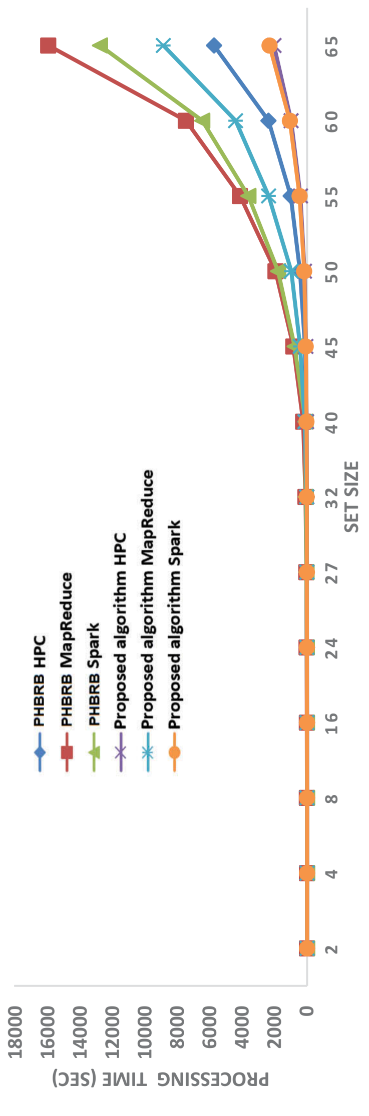


Figure 6. Elapsed time for powerset generation by PHBRB and proposed algorithm.

doesn't support parallel processing. By contrast, the proposed algorithms improved the overall processing time of powerset generation. This enhancement is caused by two reasons. The first reason is that the proposed algorithms provide parallel execution in all phases from splitting the dataset into subsets to collecting results from different machines. This behavior is compatible with both the MapReduce and Spark platforms execution strategy. The second reason is that the proposed algorithms break all dependencies between subsets. This means that Reducer can combine total powerset with any result generated from any mapper without waiting for another task. In addition, from Figure 6, it is observed that the processing time spent by Spark is less than that elapsed by the MapReduce because of the MapReduce portions a dataset into subsets in Map phase and saves the result in the local hard disk. This approach makes MapReduce requires a lot of time and increased latency to perform input/output operations on the disk. While spark performs in-memory processing of data like HPC, it is faster than MapReduce as there is no time required in moving the data in and out of the disk.

In Figure 6, it is observed that the overall Spark processing time is the nearest to HPC when applying the proposed algorithm. This is because Spark run all tasks concurrently in-memory among different machines in Hadoop cluster. In addition, the information retrieval in Spark in Hadoop cluster is faster than that with HPC. This is because the memory used by every machine in Hadoop cluster is smaller than the shared memory used by HPC. In fact, the time spent to retrieve big Data from huge shared memory is larger than the retrieving time of the same data from distributed memory over Hadoop cluster. However, Spark still spent the time to manage all machines in the cluster and in the network connection.

Discussion

In the previous section, the performance of the proposed algorithm is evaluated and compared with the PHBRB algorithm, considering different platforms. From the experimental results, several key points influence the overall performance of different approaches.

Data Transfer

MapReduce consumes time in mapping phase to send tasks and data into mappers' machines through the network. Similarly, Spark spends time to distribute A Resilient Distributed Dataset over cluster nodes. Therefore, the time spent in transferring data by both MapReduce and Spark influences the overall performance of MapReduce and Spark against HPC. HPC on the other hands runs all tasks immediately and uses internal bus speed for data transfer between CPU and memory. Figure 7 shows the amount of data transferred through the network among different machines in the cluster corresponding to different dataset sizes. In HPC, an optimized methodology is used to keep data moving inside a single machine and avoid data transfer through the network. The maximum data needed to be moved inside the HPC is 150 MB when the dataset contains 65 elements. On the other hand, the data transferring rate for big data over the network is a clear problem in Spark and MapReduce platforms. This huge amount of data would affect the overall performance because data communication between cluster nodes causes communication time delay.

Memory Usage

In the Spark system, the memory cluster should be at least equal to the large amount of data you need to process. Therefore, to process big data, the MapReduce will definitely be the cheaper option since hard disk space comes at a much lower rate than memory space. So, using disk storage as a main storage impacted the total performance of data processing because disk I/O is one of the slowest data transfer rate and hadoop mainly use it as a main storage in its operations. So, the Spark's benchmarks are more cost-effective since less hardware can perform the same tasks much faster. Indeed, Spark tries to load as much information as possible into memory to speed up calculations and can achieve better memory and CPU utilization. Spark dedicated all CPUs to be working on the actual computations rather than read/write operations on the disk.

On the other hand, one of the key goals of the HPC architecture is to deliver performance improvements and innovative functionality faster than competing technologies over the long run. Meanwhile, the use of open, blade-based components allows the HPC architecture to incorporate technology enhancements very quickly and tightly coupled intelligent software programs combine to deliver overall performance gains far greater than those of individual elements. A dedicated high-speed interconnection from the storage array delivers data to memory as quickly as each disk can stream. Compressed data is cached in-memory using a smart algorithm that keeps commonly accessed data in the memory instead of requiring a disk access.

Table 3 illustrates the memory usage for different platforms: MapReduce, Spark, and HPC. The total amount of memory allocated in HPC to process dataset size of 55 is 128 GB. This means the sub-powerset generated is stored in memory. So, when needed to find the complete powerset from huge memory, the retrieval data from huge memory will need complex operation from HPC and time is wasted to find target data. Spark uses a different methodology by storing sub-powersets in memory among different cluster nodes. This approach makes the retrieval sub-powerset easy and faster because the maximum memory size in each machine is 2.5 GB that is needed to store the result of the dataset that contains 55 operation elements. So, when Spark driver needs to collect sub-powersets from different nodes, the

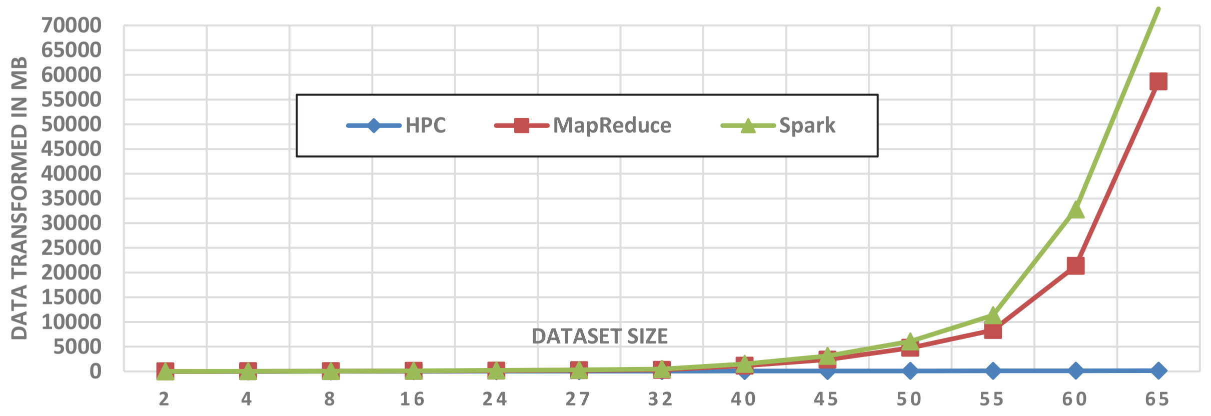


Figure 7. Volume of data during generation, data transferred among machines in the cluster corresponding to each testing data size.

Table 3. Memory usage by different platforms.

Dataset size	HPC	Spark per machine
2	2 KB	128 MB
4	16 KB	128 MB
8	128 KB	128 MB
16	4 MB	128 MB
24	512 MB	128 MB
27	1.5 GB	128 MB
32	3 GB	128 MB
40	8 GB	256 MB
45	24 GB	512 MB
50	46 GB	1 GB
55	128 GB	2.25 GB

worker nodes retrieve result quickly and master node uses hard disks to write result only.

Reliability

Reliability of distributed systems based on big data platforms is extremely important as it measures the system's ability to process data despite the continuous increase in the system load, data volume, query volume, and complexity. For example, to reliably execute long-running jobs on Hadoop clusters, the system needs to be fault-tolerant and be able to recover from failures (mainly due to machine reboots that can occur due to normal maintenance or software errors). Although Hadoop is designed to tolerate machine reboots, it uses replication task technique to avoid data node crash and big data infrastructure can process another copy (Mustafa, Ghadiri, and Tajaddini 2015; Sakr, Liu, and Fayoumi 2013). Also, it provides high availability for Name node to save the whole infrastructure in cases when name node goes faulty (Wang et al. 2016)

On the other hand, the reliability of HPC is an ability of processes to continue operating well or with minimal harm. However, unlike big data technology, the data warehouse may be represented by a single machine or rack. This means, there is a single point of failure and HPC architect should implement standby HPC to solve this issue.

Ease of Administration

Parallel and distributed system models are capable of supporting complex and difficult administration processing. In Hadoop cluster, an investment in Hadoop requires an investment in infrastructure in addition to the team responsible for managing this cluster (Naghshineh et al. 2009). The management process of Hadoop cluster is very difficult because there are a lot of management tasks more than just managing Hadoop. The management team should manage provisioning to enable install Hadoop over cluster after OS provisioned onto the cluster nodes. Then, it is responsible to monitor overall cluster to do health checks, notifications, and automatic corrective actions. So, any fault value for any property in Hadoop, MapReduce, and Spark would cause performance issue. By contrast, the deployment parallel and distributed tasks on a HPC are very easy because they limit the management tasks needed from the management team to handle hardware or platform failure since all components are managed by single software. Definitely, all steps are optimized for infrastructure and high performance to data processing.

Conclusion

This paper presents a parallel and distributed powerset generation using big data processing platforms. A new theory is introduced to improve the performance of powerset generation in different environments. It minimizes the processing time spent to generate powerset due to parallel execution in different phases of powerset generation. From the result, it is seen that the HPC provides a real-time data processing and generates powerset for large size dataset. Also, in HPC, monitoring and management software makes troubleshooting easy when any error occurs. However, when using HPC model, there is a problem concerned with scalability because it uses scale-up approach methodology. This problem is solved by scaling the cluster horizontally using big data processing concepts. This means the owner can upgrade a size of memory and increase number of CPUs using commodity machines without needing to replace any machine inside the Hadoop cluster.

Acknowledgments

This research was supported by Idealo GmbH. We thank our colleagues from Idealo who provided insight and expertise that greatly assisted the research.

ORCID

Gamal Attiya http://orcid.org/0000-0002-4771-9165

References

Amailef, K., and L. Jie. 2013. Ontology-supported case-based reasoning approach for intelligent m-government emergency response services. *Decision Support Systems* 55 (1):79–97. Elsevier. doi:10.1016/j.dss.2012.12.034.

Bahnasy, K., K. M. Naguib, and M. Aref. 2014. A novel case base indexing model based on power set tree. *International Journal of Computer Applications* 97 (7): 1-8.

Bova, V. V., V. V. Kureichik, and A. A. Lezhebokov. 2014. The integrated model of representation of problem-oriented knowledge in information systems. 8th International

Conference on Application of Information and Communication Technologies (AICT) IEEE, 1–4, Astana, Kazakhstan, IEEE.

Ciobanu, G., and D. Paraschiv. 2002. P system software simulator. *Fundamenta Informaticae* 49 (1):61–66.

Ciobanu, G., and G. Wenyuan. 2003. A parallel implementation of the transition P systems. *Proceedings of the MolCoNet Workshop on Membrane Computing (WMC2003)* 28 (03):169–169. Report.

Cristian, M. O., A. Costan, G. Antoniu, and M. S. P. Hernández. 2016. Spark versus flink: Understanding performance in big data analytics frameworks. IEEE International Conference on Cluster Computing (CLUSTER), 433–42, Taipei, Taiwan, IEEE.

Esfandiari, M., R. Babavalian, et al. 2014. Knowledge discovery in medicine: Current issue and future trend. *Expert Systems with Applications*. 41 (9):4434–63. doi:10.1016/j.eswa.2014.01.011.

Essa, Y. M., G. Attiya, and A. El-Sayed. March 2014. New framework for improving big data analysis using mobile agent. *The International Journal of Advanced Computer Science and Applications (IJACSA)* 05 (03):25–32.

Essa, Y. M., G. Attiya, and A. El-Sayed. 2013. Mobile agent based new framework for improving Big Data analysis. IEEE International Conference on Cloud Computing and Big Data (CloudCom-Asia 2013). FuZhou, China, December.

Gil, F. J., L. Fernández, F. Arroyo, and J. Alberto. 2008. Parallel algorithm for P systems implementation in multiprocessors. The Thirteenth International Symposium on Artificial Life and Robotics, Beppu, Japan, 2008.

Hu, H., Y. Wen, T. Chua, and X. Li. 2014. Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access* 2:652–87. doi:10.1109/ACCESS.2014.2332453.

Jiang, H., Y. Chen, Z. Qiao, T. H. Weng, and K. C. Li. 2015. Scaling up Mapreduce-based big data processing on multi-GPU systems. *Cluster Computing* 18 (1):369–83. doi:10.1007/s10586-014-0400-1.

Khalifa, S., Y. Elshater, K. Sundaravarathan, A. Bhat, P. Martin, F. Imam, D. Rope, M. Mcroberts, and C. Statchuk. 2016. The six pillars for building big data analytics ecosystems. *ACM Computing Surveys (CSUR)* 49 (2):33. doi:10.1145/2966278.

Kiron, D., R. Shockley, N. Kruschwitz, G. Finch, and Haydock. 2012. Analytics: The widening divide. *MIT Sloan Management Review* 53 (2):1.

MacArthur, P., Q. Liu, R. D. Russell, F. Mizero, M. Veeraraghavan, and J. M. Dennis. 2017. An integrated tutorial on infiniband, verbs, and MPI. *IEEE Communications Surveys & Tutorials* 19 (4):2894–926. doi:10.1109/COMST.2017.2746083.

Mustafa, A., M. Ghadiri, and A. Tajaddini. 2015. Relationship between efficiency in the traditional data envelopment analysis and possibility sets. *Computers and Industrial Engineering* 81:140–46. doi:10.1016/j.cie.2015.01.001.

Naghshineh, M., R. Ratnaparkhi, D. Dillenberger, J. R. Doran, C. Dorai, L. Anderson, G. Pacifici, J. L. Snowdon, A. Azagury, M. VanderWiele, et al. 2009. IBM research division cloud computing initiative. *IBM Journal of Research and Development* 53 (4):1. doi:10.1147/JRD.2009.5429055.

Philip, C., and C. Y. Zhang, PHBRB. 2014. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences* 275:314–47. doi: 10.1016/j.ins.2014.01.015.

Pronk, S., I. Pouya, M. Lundborg, G. Rotskoff, B. Wesén, P. M. Kasson, and E. Lindahl. 2015. Molecular simulation workflows as parallel algorithms: The execution engine of copernicus, a distributed high-performance computing platform. *Journal of Chemical Theory and Computation* 11 (6):2600–08. doi:10.1021/acs.jctc.5b00234.

Rashedi, E., H. Nezamabadi, and S. Saryazdi. 2013. A simultaneous feature adaptation and feature selection method for content-based image retrieval systems. *Knowledge-Based Systems* 39:85–94. Elsevier. doi:10.1016/j.knosys.2012.10.011.

Sahakyan, H. 2014. Essential points of the n-cube subset partitioning characterization. *Discrete Applied Mathematics* 163:205–13. Elsevier. doi:10.1016/j.dam.2013.07.015.

Sakr, S., A. Liu, and A. G. Fayoumi. 2013. The family of MapReduce and large-scale data processing systems. *ACM Computing Surveys (CSUR)* 46 (1):11–37. doi:10.1145/2522968.2522979.

Salah, S., R. Akbarinia, and F. Masseglia. 2015. Optimizing the data-process relationship for fast mining of frequent itemsets in mapreduce. International Workshop on Machine Learning and Data Mining in Pattern Recognition, 217–31. doi:10.3348/kjr.2015.16.1.217, Hamburg, Germany.

Seol, H., W.Jeong, and et al. 2013. Reduction of association rules for big datasets in socially-aware computing. IEEE 16th International Conference on Computational Science and Engineering (CSE), 949–56, Sydney, NSW, Australia, IEEE.

Spolaôr, E.A.Cherman, and et al. 2013. ReliefF for multi-label feature selection. Brazilian Conference on Intelligent Systems, 6–11. IEEE. doi:10.1177/1753193413492914, Fortaleza, Brazil, IEEE.

Vavliakis K. N., A. L. Symeonidis, and P. A. Mitkas. 2014. Event identification in web social media through named entity recognition and topic modeling. *Data & Knowledge Engineering* 88:1–24. IEEE. doi:10.1016/j.datak.2013.08.006.

Wang, H., Z. Xu, H. Fujita, and S. Liu. 2016. Towards felicitous decision making: An overview on challenges and trends of big data. *Information Sciences* 367:747–65. doi:10.1016/j.ins.2016.07.007.

Wu, X., X. Zhu, G. Q. Wu, and W. Ding. 2014. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering* 26 (1):97–107.

Zhang, L. Tianrui, and Y. Pan. 2012. Parallel rough set based knowledge acquisition using mapReduce from big data. Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, 20–27, Beijing, China, ACM.

Zhou, and et al. 2017. A model for hidden behavior prediction of complex systems based on belief rule base and power set. IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 84, no.9, 1649 - 1655, 2017, IEEE.