MapReduce Design for Process Discovery using Passages

Il-Ha Park, Yong-Shin Kang

Research Institute of Sustainable Manufacturing System, Korea Institute of Industrial Technology, Cheonan, Chungcheongnam-do 31056, Korea
Department of Systems Management Engineering, Sungkyunkwan University, 2066 Seobu-ro, Jangan-gu, Suwon, Gyeonggi-do 16419, Korea

Corresponding Author: yskang7867@gmail.com

Abstract

This study proposed a MapReduce design for a passage-based distributed process discovery algorithm, which can discover a process model by analyzing stored event logs depending on the progress of the process. This study can be used for the diagnosis, variation measurement, and improvement of the process by discovering the process model in a smart factory environment with an autonomously changing process.

Keywords: process mining, mapreduce, big data

1. Introduction

The concept of smart factories is gaining significance all over the world to cope with intensifying global manufacturing competition and to maintain future competitiveness. In the future, the smart factory will evolve into a self-adaptive factory where all objects exchange information by connecting to each other. The physical world and the cyber world will organically converge and communicate, and objects by themselves will be able to recognize and determine situations (1–3).

When the future self-adaptive smart factory is implemented, manufacturing processes will change depending on the situation. In addition, it is expected that variations in the process will increase, and the traceability and process management of the manufacturing sites will become more and more difficult and complicated (4). In particular, measuring and understanding variations in the manufacturing process can be a great help in understanding the current performance level. However, if it is difficult to measure the variation, it is also difficult to identify the current performance level and to define the optimized process model.

A process mining technique can be effectively utilized in such an environment where various processes exist. Process mining is a series of activities that discover a process model from the logs accumulated in the operational database, check conformance between the predefined process model and the actually executed process logs, and enhance the process model by identifying bottlenecks or anomalies (5). By applying a process mining technique, we can analyze a process that has changed and has been executed depending on the situation, determine the optimal process from the overall perspective, and enhance self-adaptive rules.

A large amount of IoT data and manufacturing execution data will be generated and stored in big data storage in the smart factory environment. To discover processes from massive data, existing process mining techniques must be implemented with big data parallel processing techniques such as MapReduce. With these in mind, this study proposes a method to discover a parallel process model from the large-scale process logs based on the assumption of a smart factory with a big data system in place. Of many process discovery algorithms, this study aims to design the passage-based distributed process discovery method proposed by Aalst et al. (6) with MapReduce, which is a parallel processing programming model.

2. Background

2.1 Related Works

The trend of applying big data technology to process
mining has been emphasized by many work (7, 8). Requieg et al. (9) suggested two-steps approach to perform events correlation discovery using a MapReduce framework, and developed the method partitioning event logs at the reducer to reduce workload.

Evermann (10) has applied the MapReduce framework to alpha algorithm and heuristics miner algorithm, and evaluated the scalability of the algorithms by calculating the total job exit time based on the performance of task nodes.

2.2 Process Discovery using Passage

The passage-based process discovery (6) divides the log into sublogs, extracts net fragments from all sublogs, and then combines all net fragments into a single entire system net instead of discovering the process by analyzing the entire log at once.

The following is an example proposed by Aalst et al. (6). Let us assume an event log \( L = \{(a, b, c, d)^{40}, (b, a, c, d)^{35}, (a, b, c, e)^{30}, (b, a, c, e)^{25}, (a, b, x, d)^{1}, (a, b, e)^{1}\} \). There are a total of 132 cases.

STEP 1: First, add artificial nodes, that is, start (\( \top \)) and end (\( \bot \)), to each trace as follows:
\[
\begin{align*}
L &= \{(\top, a, b, c, d, \bot)^{40}, (\top, b, a, c, d, \bot)^{35}, (\top, a, b, c, e, \bot)^{30}, (\top, a, c, e, \bot)^{25}, (\top, a, b, x, d, \bot)^{1}, (\top, a, b, e, \bot)^{1}\}
\end{align*}
\]

STEP 2: Construct the causal structure from the above event log. If we look at \((\top, a, b, c, d, \bot)^{1}\) and \((\top, a, b, e, \bot)^{1}\) that previously occurred once, \( x \) activity occurred only once while other activities occurred more than 50 times, so exclude \( x \). A direct dependency between \( b \) and \( e \) occurs only once; thus, exclude it for the same reason. In this way, derive the causal structure as shown in Fig. 1.

STEP 3: Generate a passage for the pairs of the same input and output on the causal structure. Four minimal passages (P1-P4) are generated in the example. That is, the causal structure is vertically cut.

\[
P1 = \{(\top), \{a, b\}\}
P2 = \{(a, b), \{c\}\}
P3 = \{\{c\}, \{d, e\}\}
P4 = \{(d, e), \{\bot\}\}
\]

STEP 4: Generate four sublogs for each passage to generate the sublogs that contain passages within the event log \( L \). L1 indicates a sublog containing P1 in \( L \).
\[
L1 = <(\top, a, b), (\top, b, a)>
L2 = <(a, b, c), (b, a, c), (a, b)>
L3 = <(c, d), (c, e), (d), (e)>
L4 = <(d, \bot), (e, \bot)>
\]

STEP 5: Now, a Petri net containing place and transition for each sublog can be found. For \( a, b \) or \( a \) occurs after \( \top \); thus \( a, b \) are AND-Split. For \( c \) after \( a, b \) or \( b \) after \( a \), occurs; thus, \( a \) and \( b \) are merged with AND-Join in \( c \). In the case of \( d \) and \( e \), since only one of \( d \) and \( e \) is executed after \( c \) occurs, and only one of \( d \) and \( e \) is executed before \( \bot \), \( c \) is branched out into \( d \) and \( e \) with OR-Split, and they are merged with OR-Join. Fig. 2 shows the net fragment for each passage discovered by the rules described above.

\[
\begin{align*}
\text{Fig. 2. Net fragments}
\end{align*}
\]

STEP 6: The final step is to combine the four net fragments shown in Fig. 2 into a single entire net system, as shown in Fig. 3.

\[
\begin{align*}
\text{Fig. 3. Net fragments}
\end{align*}
\]
3. MapReduce for Passage-Based Process Discovery

This section describes how to design a passage-based process discovery with a three-step MapReduce. It is assumed that the process log consists of objectID, readPoint, and timestamp, as shown in Table 1. These correspond to caseID, activity, and timestamp, respectively.

Table 1. Example of process log.

<table>
<thead>
<tr>
<th>objectID</th>
<th>readPoint</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>d</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Fig. 4 shows the entire MapReduce flow consisting of three steps, and the input and output of Map and Reduce of each step. The following is a detailed description of each step.

The pairs (readPoint, timestamp) are extracted using the objectID as key in Map1 of the first MapReduce. An example is shown in Fig. 5.

1: <(a,1) (b,4) (c,7) (d,9) (e,12) (f,15) (b,16) (d,18) (c,23) (e,26) (g,29)>
2: <(a,2) (b,5) (d,10) (c,24) (e,27) (g,30)>
3: <(a,3) (b,6) (c,8) (d,11) (e,13) (f,14) (b,17) (c,19) (d,20) (e,21) (f,22) (b,25) (d,28) (c,31) (e,32) (g,33)>

Fig. 5. Output of Map1

Next, the start and end nodes are added for each case in Reduce1, and the readPoint pair that is in a direct follow relationship between each readPoint is extracted. The key of the output is the readPoint name. ‘i’ is added after the readPoint name for the input node, and ‘o’ is added after the readPoint name for the output node. The value is the log number, which is the key for collecting all readPoint pairs in the next MapReduce and is a meaningless value that is not used in the calculation. This is summarized in Fig. 6.

1: <((start,ao),1) ((ai,bo),1) ((bi,co),1) ((ci,do),1) ((di,eo),1) ((ei,fo),1) ((fi,bo),1) ((bi,do),1) ((di,co),1) ((ci,eo),1) ((ei,go),1) ((gi,endo),1)>
2: <((start,a),1) ((ai,bo),1) ((bi,do),1) ((di,co),1) ((ci,eo),1) ((ei,go),1) ((gi,endo),1)>
3: <((start,a),1) ((ai,bo),1) ((bi,co),1) ((ci,do),1) ((di,co),1) ((ei,fo),1) ((fi,bo),1) ((bi,co),1) ((ci,do),1) ((di,co),1) ((ei,fo),1) ((fi,bo),1) ((bi,do),1) ((di,co),1) ((ci,eo),1) ((ei,go),1) ((gi,endo),1)>

Fig. 6. Output of Reduce1

All activity pairs are collected using logNumber 1 as a...
key, and duplicates are removed in Map2 of the second MapReduce, as shown in Fig. 7. Here, the causal structure is expressed by using the output of the map shown in Fig. 8.

![Fig. 7. Output of Map2](image)

Minimal passages are extracted in the Reduce2 step. For this, this step searches for values with the same input readPoint or output readPoint and assigns the same number, that is, the passage number.

![Fig. 8. Causal structure extracted from output of Map2](image)

The input readPoint and output readPoint values are emitted using the passage number as a key in Map3 of the third MapReduce, as shown in Fig. 10.

![Fig. 9. Numbering of minimal passages](image)

Minimal passages are extracted in the Reduce3 step, as shown in Fig. 11. The log that contains each passage is searched for from the raw log, and the sublogs for each passage are generated as shown in Fig. 12.

![Fig. 10. Minimal passages extracted from Reduce2](image)

### 4. Conclusions

This study proposed a design method to implement a process mining technique that can be used in a self-adaptive smart factory to be constructed in the near future with MapReduce, which is a parallel processing technology. First, we reviewed a passage-based process discovery method and designed a three-step MapReduce function. In addition, we described the principle of the MapReduce design by showing an example of the output of each step. This study is significant in that it opens the possibility to improve the computation performance by designing the
proposed algorithm with MapReduce, which is a parallel processing programming model.

Future research is required to verify the performance using Hadoop MapReduce, MapReduce of NoSQL databases, and Spark, which are technologies to implement the MapReduce design presented in this study.

**Acknowledgment**

This work was supported by the Technology Innovation Program (10080296, Development of Advanced Operation Management System for Smart Factory Based on Clean Energy) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea) and the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2016R1A6A3A11930205).

This work was supported by the Common Demand base Production Technology Development Program (Machinability diagnosis and control system based on deep learning for self-optimized manufacturing system) funded by the Ministry of Strategy and Finance (MI, Korea)

**References**


