model at manufacturing [15], that is universal to be implemented to other industries as well. The terms used in the algorithm are defined based on Fig. 5. The explanation of Fig. 5 is interrelated with Fig. 2 and Fig. 3 in the previous subsection.

Algorithm 1 shows the proposed approach to detect counterfeit and missing items in the RFID event stream. We only focus on the operation to detect the uncertain event between the production events and the packaging events in the event stream. The algorithm is as follows;

ALGORITHM 1: Uncertain Detection Algorithm (UDA)

```
INPUT: xmlEvent
1.  BEGIN
    //Read XML as stream
2.  WHILE(xmlStreamReader ← hasNext(dalam ni apa))
3.      IF (tagType==xmlEvent ← START_ELEMENT)
4.          SWITCH (xmlStreamReader)
5.              CASE "eventOccuredAt":
6.                  update temporaryEventOccuredAt =
                    eventOccuredAt
7.                  set internal_clock_time = eventOccuredAt
8.                  BREAK
9.              CASE "element":
10.                 IF (epcEvent == xmlEvent ←
                    START_ELEMENT)
11.                     Push resource →
                        temporaryEventOccuredAt in
                        temporaryConcurrentHashMap as key →
                        value
12.                     BREAK
13.                 END IF
14.         END SWITCH
15.     ENDIF
16.     ELSE IF (tagType==xmlEvent → END_ELEMENT)
            IF (tagType == ObjectEvent)
            //Append temporaryConcurrentHashMap in new
            thread
17.         IF (PRODUCTION_TID ←
            concurrentHashMap(eventOccuredAt)
18.             FOR(entry ← PRODUCTION_TID)
19.                 IF(entry(eventOccuredAt))
20.                     PRODUCTION_TID++
21.                 END IF
22.             END FOR
23.         END IF
24.         LOST_ITEM++
25.         REMOVE ALL LIST OF PRODUCTION_TID after
            t-winc
26.     ENDIF
27.     CLEAR temporaryConcurrentHashMap
28.     CLEAR temporaryEventOccuredAt
            ELSE IF tagType==AggreagtionEvent
            //Append temporaryConcurrentHashMap in new
            thread
29.         IF (PACKAGING_TID ←
            concurrentHashMap(eventOccuredTime)
30.             FOR(entry ← PACKAGING_TID)
31.                 IF(entry == eventOccuredAt)
32.                   keyset ← entry
33.                 END IF
34.             END FOR
35.         END IF
36.         COUNTERFEIT_ITEM++
37.         REMOVE ALL LIST OF PACKAGING_TID after t-
            winf
38.     END IF
39.     CLEAR temporaryConcurrentHashMap
40.     CLEAR temporaryEventOccuredAt
41.     END IF
42. END WHILE
43. END
```

In this algorithm, the case 'eventOccuredAt' is to update the internal clock. While case 'element' reads the epc and adds them into the temporaryConcurrentHashmap.

Based in Algorithm 1, at line 1 to line 3, read the incoming event as streaming data. At line 3, if it is START_ELEMENT, store the occurrence time of an event in a temporary string variable (line 5 to line 8) and the elements of events in the temporaryConcurrentHashMap (line 9 to line 12). At line 16, if the tag is END_ELEMENT?, and if the event is from the production area (ObjectEvent), in a separate thread, we pass the temporary event's occurrence time and temporaryConcurrentHashMap into the thread (line 17 to line 26). After passing the resource, the clear eventOccurredAt, temporaryConcurrentHashMap and started reading the next event (line 28 and line 29). In line 30, if the event is from packaging area (AggregationEvent), in a separate thread, we pass the temporary event's occurrence time, and temporaryConcurrentHashMap into the thread (line 31 to line 40). After passing the resource, the clear eventOccurredAt, temporaryConcurrentHashMap and started reading the next event (line 41 and line 42).

This is the section where we create a thread pool with a fixed number of threads. A separate task is created for every event in the event stream and is submitted it to the thread pool.

In line 18, if the event just now was a production event, in a separate thread, merge the PRODUCTION_IDs to concurrentHashMap. In the thread pool, start the processing function to auto-remove and push missing items (line 19 to line 26). If there is a temporary entry from production, for which event occurred time in addition with active window detection where length is less than or equal to time extracted from the current event, report the PACKAGING_TID as missing item detected in production.

At line 31 to line 39, if the event just now was a packaging event, in a separate thread, merge the PACKAGING_TIDs into concurrentHashMap (line 31). In thread pool line 32 to line 33, if a pair of resource or element is found between production's thread and packaging's thread, it is added to the hashmap of matched items. If there is a temporary entry from packaging, for which event occurred time in addition with active window detection where length is less than or equal to time extracted from the current event, report the PACKAGING_TID as counterfeit detected in packaging. At the end of the process, the main thread checks if there are any threads still running, the process will hold a bit until all the process terminates itself.

III. Results and Discussion

In order to prove the efficiency of the proposed approach, performance analysis was conducted and compared with the existing approaches and tools. For the computer environment, Intel i3 processor and eight-gigabyte memory were used, and the operating system is Windows 7. The modules for CEP were implemented in Java. The dataset used in the experiment is presented as a time-varying stream of events that represents an observation of the actual counterfeiting and theft detection scenarios from the pharmaceutical company recorded by the sensor [15].

The performance evaluation of three processing approaches, proposed approach, Instans [50], ESPER [16], NFA-HTS [40] and LCA [43]. A slight modification has been

made to these algorithms to solve the problems. We measured mainly execution time, memory consumption and detection accuracy according to the event stream scale with 8% uncertain events. These measurements are defined as follows;

- Execution time: the time required to detect uncertain events according to the event stream scale.
- Memory consumption: memory consumed when detecting uncertain events in the event stream according to the event stream scale.
- Detection Accuracy: measures the accuracy of uncertain events according to the event stream scale.

The system execution time or throughput is the time it takes to load and analyse an event stream. It also includes the time it takes to load the output into a consistent format and parse it, and time it takes to write the results to disk. While memory consumption is the amount of memory required to load and analyse while processing an event stream. Detection accuracy is the total accuracy detected over the 8% uncertain event in the event stream. For robustness, we performed ten (10) independent runs for each experiment, and we report the median values. The number of event streams scale is varied from 1000 to 5000 with an increment of times 1000 for each scale.

A. Analysis Based on Detection Accuracy

In this section, extensive experiments were performed to find the detection accuracy of the proposed approach, Instans and ESPER were studied under varying event stream scale. Fig. 6 illustrates the performance of the proposed approach, Instans and ESPER with 8% uncertain events under varying event stream scale. Fig. 6 shows that, UDA and NFA-HTS gets 100% accurate. Instans and ESPER also maintain to get accurate detection at every event stream scale. For Instans, Rete implementations are non-distributed implementation, yet efficient in term of detection accuracy.

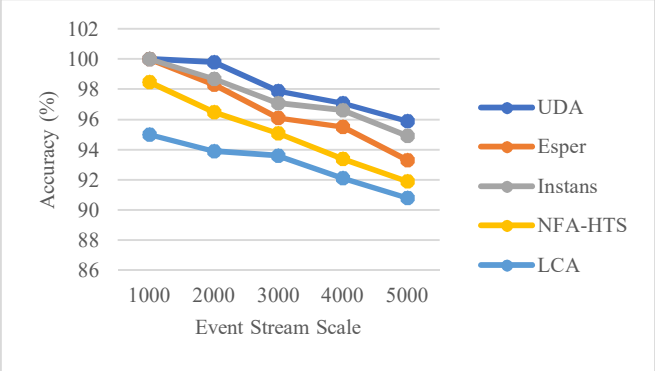

Fig. 6 Detection accuracy percentage under different event stream scale

ESPER is an open source Java-based event stream processing engine that analyzes a set of events and draws conclusions from the event stream. It provides a language called Event Processing Language (EPL) that implements and extends the SQL standard. In Instans, the evaluation of SPARQL (events) subscriptions is sequential. While in the proposed approach (UDA), the evaluation is multithreaded.

B. Analysis Based on Execution Time

Extensive experiments have been conducted in this section to see the results of reduced execution times. Fig. 7 below shows the execution time of the proposed approach using different event stream scale sizes. The execution times of the three approaches to the event streams range 1000 to 5000 events. The UDA took the least time, followed by NFA-HTS. While Instans took the longest time to process the event stream followed by LCA. This is because Instans has to focus on continuously processing RDF event streams while employing the Rete algorithm and propagating data through a matching network. By using Rete, the event matches are produced as soon as all the conditions of SPARQL graph patterns are matched. While the LCA algorithm uses a simple NFA structure to determine potential matches. In other words, events will be buffered until they find a match.

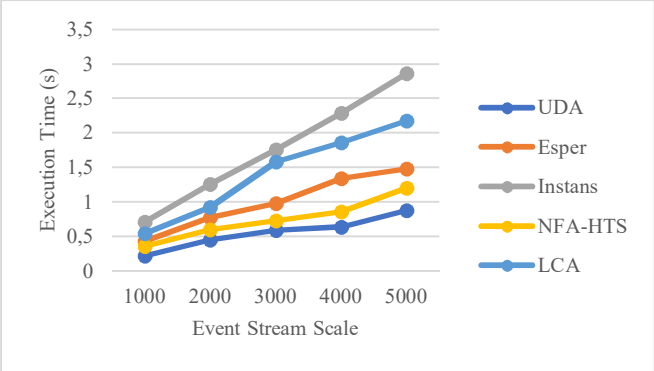

Fig. 7 The execution time of the proposed approach under different event stream scale

While using the UDA algorithm, the event has to match the pattern as soon as the event is merged in the thread pool. At the end of the process, in this approach, the main aim is that if there are any threads still running, the process will hold a bit until all the process terminates automatically. The graph also indirectly shows that by using ConcurrentHashMap and thread pool technique, it requires less time to process events compared to NFA-HTS. NFA-HTS also shares some advantages by using hash table structure. It enables easy retrieval and reducing computation operations based on storage technologies of hash table structure. ESPER is an engine primarily relying on state machines. For ESPER, we do not have a very detailed architectural diagram because the community does not release such materials to the public domain. The code is open source and the API's to use the engine is sufficiently documented.

C. Analysis Based on Memory Consumption

In this section, the proposed approach is compared with Instans, ESPER, NFA-HTS and LCA in terms of memory consumption under different event stream scales. As depicted in Fig. 8, UDA shows the least consumption and slightly better than NFA-HTS compared to the Instans, ESPER and LCA. While Instans and ESPER show higher memory consumption, especially when the event stream scale is massive. This is because, by using the thread pool, we do not have to create, manage, schedule and terminate the thread. Thread pool help mitigate performance issues by reducing the number of threads required and managing their lifecycle.

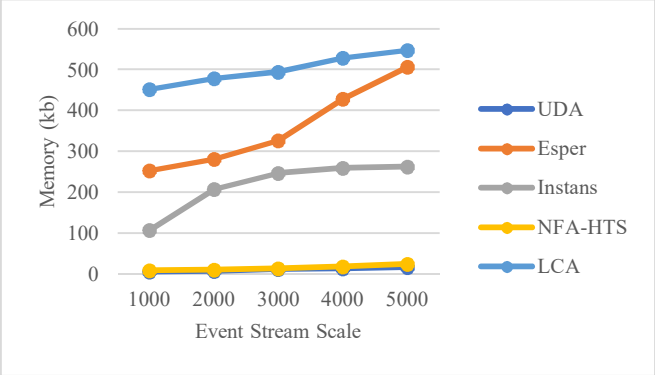

Fig. 8 Memory consumption under different event stream scale

Essentially, thread pool reuses its threads to perform the work pattern detection until it is completed. In non-technical explanation, the threads continue to be in the pool till they may be required, after which they execute the task and return the pool to be reused later so that it can pick up the next available task. This mechanism is beneficial in systems or engines that execute a large number of small tasks or processes. Besides creating threads, destroying a thread has set up and torn down costs (it takes CPU time). While in LCA, the engine continuously buffers until they find a match. Same as Instans, the engine is in charge of continuously evaluating the incoming data and storing intermediate results. At the same time, ESPER exploits in-memory processing to address the requirements of applications that analyze a high volume of event stream and promptly react to events by applying complex computations.

IV. Conclusion

In this research, the issue of uncertain event detection has been studied, and a brand new approach based on hashing and thread pool technique was proposed. The overall performance of the proposed approach is compared with the existing approaches. The results show that the performance of the proposed approach was better than the opposite approaches in terms of detection time and memory usage. Moreover, this research additionally demonstrated that, with the aid of using thread pool technique, the proposed approach performs quicker than the opposites in terms of the efficiency of the experiments. Despite the convenience of use, the thread pool has the subsequent limitations when compared to manually handling the threads. With thread pool, we do not have control over the state and priority of the thread. We additionally cannot provide a solid identification to the thread and maintain monitoring it. The technique may be inefficient whilst there is a excessice demand for the thread pool.

Synchronization overhead is one of the reasons that lead to inaccurate detections. Even though the detection accuracy deteriorates when the event stream scale increases, the proposed approach processes at the optimum rate when the event stream is at a moderate scale. For future work, we plan to expand our research on the relationship between pool size and the number of threads can be set dynamically.

Acknowledgement

This research is sponsored by Universiti Tun Hussein Onn Malaysia under Fundamental Research Grant Scheme Vot 1611 and the Ministry of Higher Education through MyBrain scholarship.

References

[1] Guthrie, John, Sarah Todd, and Jeffrey Alstete. "Inside advice on educating managers for preventing employee theft." International Journal of Retail & Distribution Management (2006).

[2] Singh, Mohan, Smriti Sachan, Akansha Singh, and Krishna Kant Singh. "Internet of Things in pharma industry: possibilities and challenges." In Emergence of Pharmaceutical Industry Growth with Industrial IoT Approach, pp. 195-216. Academic Press, 2020.

[3] Siddiqa, A., Hashem, I.A.T., Yaqoob, I., Marjani, M., Shamshirband, S., Gani, A. and Nasaruddin, F., 2016. A survey of big data management: Taxonomy and state-of-the-art. Journal of Network and Computer Applications, 71, pp.151-166.

[4] Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Kamp, M. and Mock, M., 2017. Issues in complex event processing: Status and prospects in the big data era. Journal of Systems and Software, 127, pp.217-236.

[5] Lan, L., Shi, R., Wang, B., Zhang, L. and Jiang, N., 2019. A universal complex event processing mechanism based on edge computing for internet of things real-time monitoring. IEEE Access, 7, pp.101865-101878.

[6] Correcher, J.F., Alonso, M.T., Parreño, F. and Alvarez-Valdés, R., 2017. Solving a large multicontainer loading problem in the car manufacturing industry. Computers & Operations Research, 82, pp.139-152.

[7] Bhat, S. and Krishnamurthy, A., 2016. Interactive effects of seasonal-demand characteristics on manufacturing systems. International Journal of Production Research, 54(10), pp.2951-2964.

[8] Wang, Y., Gao, H. and Chen, G., 2018. Predictive complex event processing based on evolving Bayesian networks. Pattern Recognition Letters, 105, pp.207-216.

[9] Muzammal, M., Gohar, M., Rahman, A.U., Qu, Q., Ahmad, A. and Jeon, G., 2017. Trajectory mining using uncertain sensor data. IEEE Access, 6, pp.4895-4903.

[10] Lee, O.J. and Jung, J.E., 2017. Sequence clustering-based automated rule generation for adaptive complex event processing. Future Generation Computer Systems, 66, pp.100-109.

[11] Cugola, G. and Margara, A., 2012. Low latency complex event processing on parallel hardware. Journal of Parallel and Distributed Computing, 72(2), pp.205-218.

[12] Hewa Raga Munige, T., 2016. Real time stream processing for Internet of Things and sensing environments (Doctoral dissertation, Colorado State University).

[13] Hallé, S., 2017. From complex event processing to simple event processing. arXiv preprint arXiv:1702.08051.

[14] Rinne, M., Solanki, M. and Nuutila, E., 2016, June. RFID-based logistics monitoring with semantics-driven event processing. In Proceedings of the 10th ACM international conference on distributed and event-based systems (pp. 238-245).

[15] Rinne, M., Nuutila, E. and Törmä, S., 2012, November. INSTANS: High-performance event processing with standard RDF and SPARQL. In 11th International Semantic Web Conference ISWC (Vol. 914, pp. 101-104).

[16] "EsperTech." http://www.espertech.com/ (accessed May 29, 2019).

[17] Sarac, A., Absi, N. and Dauzere-Peres, S., 2015. Impacts of RFID technologies on supply chains: a simulation study of a three-level supply chain subject to shrinkage and delivery errors. European Journal of Industrial Engineering, 9(1), pp.27-52.

[18] Fan, T., Tao, F., Deng, S. and Li, S., 2015. Impact of RFID Technology on Supply Chain Decisions with Inventory Inaccuracies. International Journal of Production Economics, 159, pp.117-125.

[19] Yao, X., Zhang, J., Li, Y. and Zhang, C., 2018. Towards flexible RFID event-driven integrated manufacturing for make-to-order production. International Journal of Computer Integrated Manufacturing, 31(3), pp.228-242.

[20] Wang, F., Liu, S. and Liu, P., 2009. Complex RFID event processing. The VLDB Journal, 18(4), pp.913-931.

[21] Q. J. Lei, L. S. Bo, and C. J. Kun, "Online Monitoring of Manufacturing Process Based on autoCEP," International Journal of Online Engineering (iJOE), vol. 13, no. 06, pp. 22–34, Jun. 2017.
[22] Vlahakis, G., Apostolou, D. and Kopanaki, E., 2018. Enabling situation awareness with supply chain event management. Expert Systems with Applications, 93, pp.86-103.
[23] Jia, X., Wenming, Y. and Dong, W., 2009, November. Complex event processing model for distributed RFID network. In Proceedings of the 2nd international Conference on interaction Sciences: information Technology, Culture and Human (pp. 1219-1222).
[24] Brunelli, D., Gallo, G. and Benini, L., 2016, September. Sensormind: virtual sensing and complex event detection for Internet of Things. In International Conference on Applications in Electronics Pervading Industry, Environment and Society (pp. 75-83). Springer, Cham.
[25] Zhang, Y. and Sheng, V.S., 2018. Fog-enabled Event Processing Based on IoT Resource Models. IEEE Transactions on Knowledge and Data Engineering, 31(9), pp.1707-1721.
[26] Wang, Y., Zheng, L., Hu, Y. and Fan, W., 2018, December. Multi-source heterogeneous data collection and fusion for manufacturing workshop based on complex event processing. In Proceedings of the 48th International Conference on Computers & Industrial Engineering (CIE), Auckland, New Zealand (pp. 2-5).
[27] Agrawal, J., Diao, Y., Gyllstrom, D. and Immerman, N., 2008, June. Efficient pattern matching over event streams. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 147-160).
[28] Demers, A.J., Gehrke, J., Panda, B., Riedewald, M., Sharma, V. and White, W.M., 2007, January. Cayuga: A General Purpose Event Monitoring System. In Cidr (Vol. 7, pp. 412-422).
[29] Brenna, L., Demers, A., Gehrke, J., Hong, M., Ossher, J., Panda, B., Riedewald, M., Thatte, M. and White, W., 2007, June. Cayuga: a high-performance event processing engine. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (pp. 1100-1102).
[30] Welbourne, E., Khoussainova, N., Letchner, J., Li, Y., Balazinska, M., Borriello, G. and Suciu, D., 2008, June. Cascadia: a system for specifying, detecting, and managing RFID events. In Proceedings of the 6th international conference on Mobile systems, applications, and services (pp. 281-294).
[31] Gillani, S., Zimmermann, A., Picard, G. and Laforest, F., 2019. A query language for semantic complex event processing: Syntax, semantics and implementation. Semantic Web, 10(1), pp.53-93.
[32] Akila, V., Govindasamy, V. and Sandosh, S., 2016, April. Complex event processing over uncertain events: Techniques, challenges, and future directions. In 2016 International Conference on Computation of Power, Energy Information and Commuincation (ICCPEIC) (pp. 204-221). IEEE.
[33] Rincé, R., Kervarc, R. and Leray, P., 2018, September. Complex event processing under uncertainty using Markov chains, constraints, and sampling. In International Joint Conference on Rules and Reasoning (pp. 147-163). Springer, Cham.
[34] Tang, L., Cao, H., Zheng, L. and Huang, N., 2015. Value-driven uncertainty-aware data processing for an RFID-enabled mixed-model assembly line. International Journal of Production Economics, 165, pp.273-281.
[35] Cugola, G., Margara, A., Matteucci, M. and Tamburrelli, G., 2015. Introducing uncertainty in complex event processing: model, implementation, and validation. Computing, 97(2), pp.103-144.
[36] Cugola, G. and Margara, A., 2012. Processing flows of information: From data stream to complex event processing. ACM Computing Surveys (CSUR), 44(3), pp.1-62.
[37] Akram, N., Siriwardene, S., Jayasinghe, M., Dayarathna, M., Perera, I., Fernando, S., Perera, S., Bandara, U. and Suhothayan, S., 2017, June. Anomaly detection of manufacturing equipment via high performance rdf data stream processing: Grand challenge. In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (pp. 280-285).
[38] Li, F., Wang, N., Gu, Y. and Chen, Z., 2016, September. Effective Privacy Preservation over Composite Events with Markov Correlations. In 2016 13th Web Information Systems and Applications Conference (WISA) (pp. 215-220). IEEE.
[39] Alevizos, E., Artikis, A., Katzouris, N., Michelioudakis, E., Paliouras, G. and Paliouras, G., 2018. The Complex Event Recognition Group. ACM SIGMOD Record, 47(2), pp.61-66.
[40] Wang, J., Liu, J., Lan, Y. and Cheng, L., 2018. An Efficient Complex Event Detection Algorithm based on NFA_HTS for Massive RFID Event Stream. Journal of Electrical Engineering and Technology, 13(2), pp.989-997.
[41] Wang, J., Lu, S., Lan, Y. and Cheng, L., 2018. An Efficient Complex Event Processing Algorithm Based on NFA-HTBTS for Massive RFID Event Stream. International Journal of Information Technologies and Systems Approach (IJITSA), 11(2), pp.18-30.
[42] Bok, K., Kim, D. and Yoo, J., 2018. Complex event processing for sensor stream data. Sensors, 18(9), p.3084.
[43] Kolchinsky, I. and Schuster, A., 2018. Efficient adaptive detection of complex event patterns. arXiv preprint arXiv:1801.08588.
[44] K. Tawsif, J. Hossen, J. Emerson Raja, M. Z. H. Jesmeen, and E. M. H. Arif, "A Review on Complex Event Processing Systems for Big Data," in Proceedings - 2018 4th International Conference on Information Retrieval and Knowledge Management: Diving into Data Sciences, CAMP 2018, Mar. 2018, pp. 2–7.
[45] Yin, S.N., Kang, H.S., Chen, Z.G. and Kim, S.R., 2016, October. Intrusion detection system based on complex event processing in RFID middleware. In Proceedings of the International Conference on Research in Adaptive and Convergent Systems (pp. 125-129).
[46] Rinne, M. and Nuutila, E., 2014, October. Constructing event processing systems of layered and heterogeneous events with SPARQL. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems" (pp. 682-699). Springer, Berlin, Heidelberg.
[47] Arbuzin, D., 2017. Real-time detection of moving crowds using spatio-temporal data streams.
[48] Stusek, M., Masek, P., Zeman, K., Kovac, D., Cika, P., Pokorny, J. and Kröpfl, F., 2016. A Novel Application of CWMP: An Operator-grade Management Platform for IoT. International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems, 5(3), pp.171-177.
[49] Jantkal, B.A. and Deshpande, S.L., 2017, August. Hybridization of B-Tree and HashMap for optimized search engine indexing. In 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon) (pp. 401-404). IEEE.
[50] Pinto, G., Torres, W., Fernandes, B., Castor, F. and Barros, R.S., 2015. A large-scale study on the usage of Java's concurrent programming constructs. Journal of Systems and Software, 106, pp.59-81.
[51] Debnath, B., Haghdoost, A., Kadav, A., Khatib, M.G. and Ungureanu, C., 2016. Revisiting hash table design for phase change memory. ACM SIGOPS Operating Systems Review, 49(2), pp.18-26.
[52] Zheng, Yajie, Dongwen Zhang, Yang Zhang, Song Guo, Yanan Liang, Mengmeng Wei, and Xin Yu. "Comparison and reconfiguration of Java hash mechanisms on parallel environment." Hebei Journal of Industrial Science and Technology (2017): 06.
[53] Wei, L., Wan, S., Guo, J. and Wong, K.K., 2017. A novel hierarchical selective ensemble classifier with bioinformatics application. Artificial intelligence in medicine, 83, pp.82-90.
[54] Du, P., Ren, J., Pan, J. and Luo, A., 2014. New cross-matching algorithm in large-scale catalogs with ThreadPool technique. Science China Physics, Mechanics and Astronomy, 57(3), pp.577-583.
[55] Bhargavi, R., 2016. Complex Event Processing Framework for Big Data Applications. In Data Science and Big Data Computing (pp. 41-56). Springer, Cham.