Theories of Real Time Stream Processing Architecture Using Task Forking Model

Shiladitya Munshi

Research Student, Department of Computer Science & Engineering, The Assam Kaziranga University, Jorhat, Assam 785006, India
munshi.shiladitya@gmail.com

Sajal Saha

Professor and Head, Department of Computer Science & Engineering, Adamas University, Barasat, West Bengal, 700126, India
sajalkrsaha@gmail.com

Radha Tamal Goswami

Director, Techno International Newtown, Newtown Megacity, West Bengal, 700156, India
rtgoswami@tict.edu.in

Abstract

Present study reports the theoretical aspects of big data stream processing architecture and it's performance metric identification, performance optimization and related experiments. The proposed architecture is considered as a complex directed graph model with various real time computational elements as nodes and big data tuples as edges, forming a real time topology. The notions of hard time deadline bound computation on streaming big data tuple along with minimum performance guarantee of processing every tuple have been introduced in the present research. Time bound computation issues in the real time stream computing architecture have been improved by optimizing time deadline management through task forking models. An algorithm for optimization of throughput has been reported and the performance metrics of the proposed system has also been identified for proper analysis on the basis of queuing theories by expressing it through appropriate Kendalls notation. Experimental results, at the end, report considerable improvement of performance of the architecture by applying optimization algorithms on a standard dataset.

Keywords: **Stream Processing, Real Time Processing, Task Forking Model, Queuing Theory, Kendalls Notation.**

1. Introduction

In recent years, a new type of data-centric applications has emerged into the market where the data is best considered not to be as persistent objects rather, they are modeled as transient streams. Moreover, these data streams are generated at great speed and decision are to be taken in real time based on the changing state of the data streams [1]. Hence these new generations of data-centric applications demand real time analytics over spurious streams of data having enormous generation speed.

Current state of the art data processing infrastructures like Apache Storm and Apache Flink, offer excellent low latency results on processing high velocity streams of messages, but the elements of real time computations and queuing theory or sophisticated resource allocations are hard to be traced in their solution architecture, as the primary focus of the current streaming data processing technologies is the low latency with binary output (either the entire stream will be processed or will be rejected), not the real time performance guarantee or the optimization of query processing results.

As mentioned in [1], the streaming data computation is not possible in traditional database management systems (DBMS) due to the fact that neither DBMS are designed for rapid and continuous storing of discrete data elements, nor they have supported the continuous queries required for processing data streams. While the virtue of approximation [2] and adaptability [3] share crucial impact in executing queries over rapid data streams, traditional DBMS are still supported by stable query plans, in contrast, for computing precise answers.

Hence there exists a fair scope of further analysis of the generic real time stream computation architecture for gaining more insights and utilizing the knowledge derived to optimize the data processing metrics. With this motivation, current study reports a theoretical foundation of stream processing in real time along with its

optimization and performance evaluations. The proposed architecture is expected to reveal various insights of streaming data processing that might augment the current advancement of real time data processing as a whole

On this background, current paper has been organized as follows:

Section 2 discusses the related research works in the domain of real time streaming data processing and section 3 introduces a generic architecture of a real time data stream processor. It elaborates the generic design requirements and proposes the basic communication model among different computational units. Section 4 next, discusses the theories of task forking model and impact of different forking strategies over the real time stream processing architecture and it's possible optimizations. While section 5 investigates the response delay with the different queuing models to introduce new performance metrics, Section 6 reports the experimental facts and figures about the proposed stream processing architecture along with optimization algorithms mentioned. Current paper ends with formal conclusion and bibliography

2. Related Works

Real time systems have been studied in great details by the researchers since last two to three decades [2–9]. These studies have mostly reviewed real time scheduling which form the basis of real time computing in distributed mode. Further, various issues in real time data computation in distributed mode has been analyzed in [10–12].

Optimizations of scheduling algorithm of Real Time Systems from various targeted perspectives have emerged as thrust areas of recent research on real tie systems. [31] has dealt this issue for energy efficient systems, whereas [32] has reported novel schedulabiity of probabilistic mixed criticality systems. Optimal scheduling of parallel real time tasks on Directed Acyclic Graph based topologies have been detailed in [33].

More targeted studies towards data stream processing have been carried out in [13–17]. Various mathematical models for data stream computations have been thoroughly discussed in [18, 19]

Apache Storm has a message passing guarantee ensuring the fact that the message has been processed fully. Storm achieves this guarantee based on a higher level abstraction technology called "Trident", which, basically, is a micro batching environment. Storm message might either be failed to be processed or be processed fully. There is no partial processing, which scopes out the probabilistic performance guarantee model of classical real time systems. Moreover, to inspect the states of message processing, Apache Storm employs small storage either in memory, or in Memcached, or in Cassandra, or in some other store, making the system to work under a global timeout (30 seconds by default) , not offering any scope of processor level time out. This absence of granular time deadline management has a huge impact in the form of absence of probabilistic performance guarantee. The Storm like at-least once processing guarantee might go well with non-crucial information processing, but real time hardware based controls those are directly related with crucial impacts like flight monitoring, space vehicle monitoring or forest fire disaster management, would certainly require more granular reliability over the processing elements. Hence even if Apache Storm or it's updated counterpart Twitter's Heron [37] exists in the market, a generic real time stream processing architecture is required for more generic version of "realtimeness" in the stream processing architecture

Recent industrial advancements in real time architecture have been manifested through Apache StormTM[20]. Research works carried out in [21–24] focus in enhancing and optimizing Storm architecture for better performance. More generic theoretical studies on real time big data processing have been reported in [25–29]. Though standard of Apache HadoopTM has evolved as a batch processing environment, still researchers have reported their studies in enabling Hadoop framework to process big data in real time [30, 34]

All the research studies reported in the preceding discussions and a concise survey [35], provide a strong base to look at the problem of processing big data in real time in the form of stream processing framework. A rudimentary study was carried out in our own work [36] to present a very basic and conceptual idea of real time stream processing architecture.

3. Real Time Stream Processing Architecture

Present section describes a basic real time stream processing architecture (parts of which has been described as RStreaming in [36]) which has its fundamental data structure as Digraph distributed over a cluster of nodes. Following subsections illustrate different aspects of this architecture and it is noteworthy to consider that task forking, and queuing models would be applied on this digraph architecture for further optimization of query processing qualities. Discussion presented in subsection 3.1 and 3.2 recapitulates and more closely discusses some ideas which had been introduced in [36]

3.1 Logical Data Model

Real time stream processing architecture concept is logically based on a directed graph known as *RTopology*. It has got two types of nodes as seen from the higher abstraction level, one for feeding the normalized external messages into the system (*RSpout*) and the other for computation on those data (*RBolt*). RBolt can exist in two

forms, namely *DeadLineBolt* and *PerformanceBolt*. Stream of messages flows through different *RBolts* within the *RTopology*.

RSpout: *RSpout* fetches data from the data source. It has a configurable *TimeDeadLine* (with a default value in milliseconds), which could be distributed throughout all the *RBolts* in line. To honor this *TimeDeadLine*, *RSpout* maintains a GlobalClock for each incoming message of the stream conventionally known as *Tuple*.

RBolt: *RBolts* can process each *Tuple* as per application requirement. In general, business logic of the application dictates the nature of the computation or the processing of the *tuple*. Additionally, *RBolt* does some additional time management and performance measure jobs to maintain the flavors of real time computing.

DeadLineBolt: It is a dedicated computational element within *RTopology* which manages the time deadline related issues. There exists a single *DeadLineBolt* per *RTopology*. The *DeadLineBolt* is always globally connected with all the *RBolts* and *RSpout* for passing a special *TimeMessages* to and fro.

PerformanceBolt: Issues with performance guarantee are managed with this type of *RBolts*. As each and every computation is bound to be carried over within a time deadline, performance measures will always fluctuate and hence it is important to be gauged in every *RBolt*. As a result, each *RBolt* pairs up with a *PerformanceBolt* to form a *RBoltPair*. While the *TimeMessages* are passed between *DeadLineBolt* and *RBoltPairs*, internally a *Performance- Bolt* communicates with a *RBolt* to ensure whether the minimum performance guarantee level has been met or not.

Logical Data Model of the real time stream processing architecture is composed of the different nodes of the *RTopology* as discussed above.

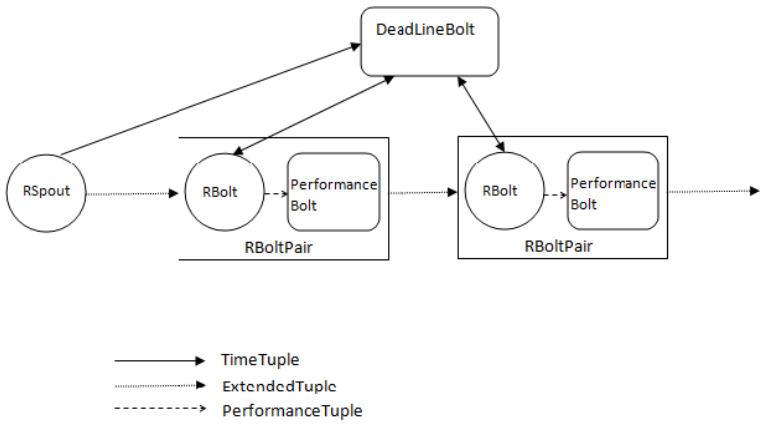


Fig. 1 Schematic Diagram of Logical Data Mode along with Different Messages being passed

As Fig.1. represents a schematic diagram of the logical model, it is evident that the success of the logical model is completely dependent on successful passing of different specially synthesize messages other than the original stream tuples. The figure denotes

- Primary computing elements like RSpout and RBolts with circle
- Auxiliary and assistive computing elements like PerformanceBolt and DeadLineBolt with rounded rectangle
- Pair of computing elements as RBoltPair with rectangle

Execution principles of the data model described above is integrated by the proper message passing among the different computational elements of the *RTopology*. There exist three types of specially designed and synthesized messages / tuples those pass through the *RTopology*. As illustrated in the Fig.1 above, they are namely (a) *ExtendedTuple*, (b) *TTuple* or *TimingTuple* and (c) *PTuple* or *PerformanceTuple*. The success of the real time streaming architecture critically trusts on the simple definition and communication strategy of these three types of Tuples. In this context, following subsection describes the different types of Tuples and their communication strategies. Fig. 1 Schematic Diagram of Logical Data Mode along with Different Messages being passed

3.2 Message Passing

Current subsection introduces different kinds of artificially synthesized messages (not the part of the original streaming messages) of the architecture. These messages carry critical information for performance guarantee and time deadline management.

Extended Tuple: Extended Tuple is passed between the RSpout and RBoltPairs. It consists of original streaming tuple followed by a PTuple which carries the preset level of performance that needs to be realized over the entire message.

TTuple: A TTuple or a Timing Tuple is the message containing the target time deadline, preset either by RSpout or RBolt for DeadLineBolt. A different version of it, ReverseTTuple carries the timing information as well, but in this case, it flows from a DeadLineBolt to a RBolt.

PTuple: A PTuple or PerformanceTuple carries the minimum preset performance level that needs to be achieved over the entire message when it flows from a RSpout to a RBoltPair. InnerPTuple, which is a bit different than PTuple, moves from the RBolt to a PerformanceBolt to carry the level of performance which has been achieved in actual by the corresponding RBolt. Moreover, there exist some more auxiliary messages as described below

TTupleRequest: TTupleRequest originates at RBolt for DeadLineBolt. In response, DeadLineBolt computes the time deadline for the corresponding RBolt and sends it back as a ReverseTTuple. The time deadline generation for a RBolt follows a specific algorithm or protocol, where the TTuple generated by RSpout for DeadLineBolt acts as an input.

The basic architecture primarily holds a pretty straightforward and static rule as shown under

Time Deadline for a RBolt= Target time deadline for the entire message / Number of RBolts downstream.

More sophisticated and dynamic rule or algorithm would further be investigated in subsequent sections of the current paper

SuccessConfirm: A *PerformanceBolt* has a reference performance level to be achieved as *PTuple* component within *ExtendedTuple*. However, it also receives the level of performance that has actually been achieved by the *RBolt* as *InnerPTuple*. *PerformanceBolt* ensures that the target performance level is actually met by generating SuccessConfirm message.

Various types of messages as described, flow within a *RTopology* to aid the computation process at *RBolts* to be completed within a fixed time deadline and to confirm minimum performance guarantee level. Fig. 2. below, reproduced from [4] for completeness of the current paper, represents an adhoc *RToplogy* with one *RSpout* and a couple of *RBoltPairs* illustrating the temporal analysis of message passing sequence.

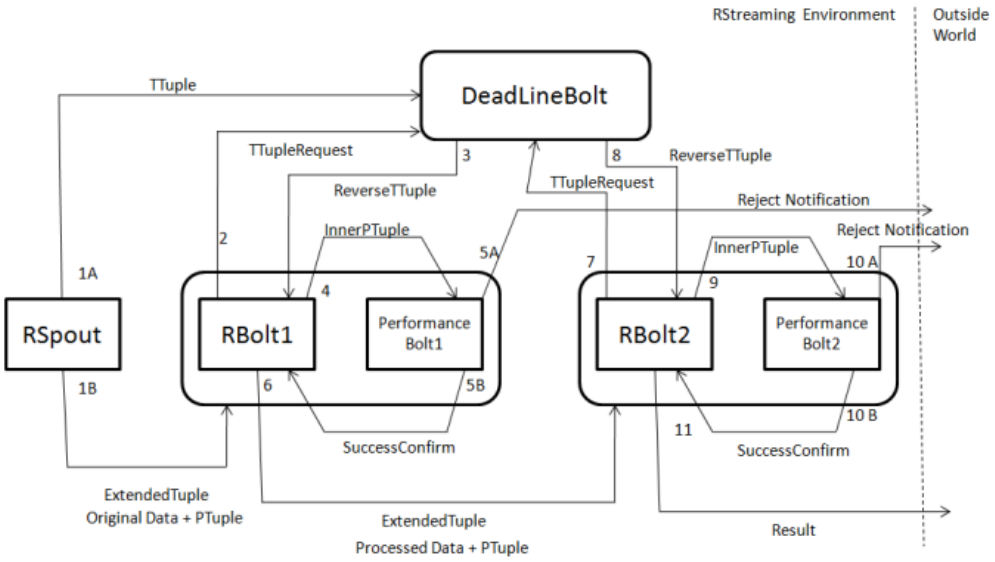


Fig. 2 Sequence of Message Passing through Rtopology

A message marked with sequence number n gets generated at a node x only after a previous message $n-1$ reaches x^{th} node. Further, any multiple messages marked with same sequence number n but with different sub

sequence order, denoted by alphabets (like 5A and 5B), are generated parallelly out of a node. The first set of messages (1A and 1B) originates out of the *RSpout* and ultimately the result is obtained through message no 11.

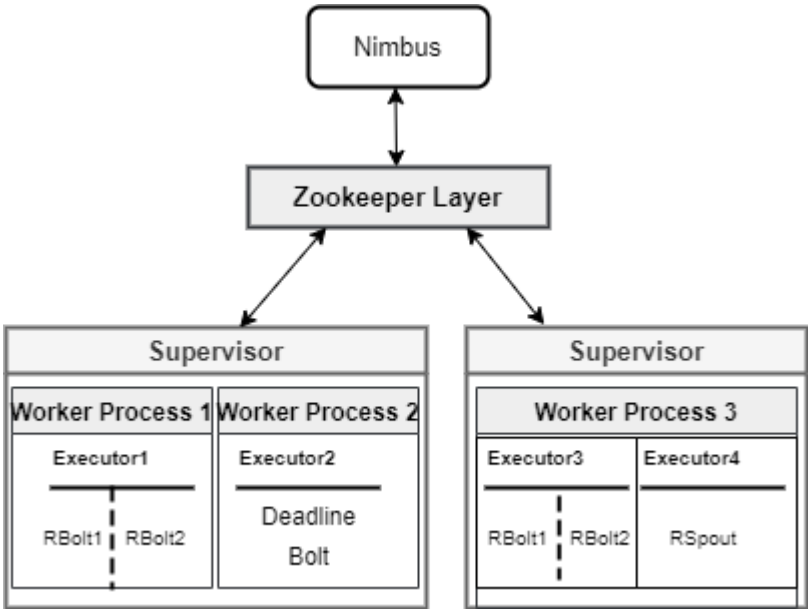


Fig. 3 A Sample Implementation of Physical Data Model Corresponding to Logical Data Model shown in Figure 2

3.3 Physical Data Model

Real time stream computation architecture is based on a distributed cluster computing approach which is inspired by the classical Apache Storm architecture. The cluster is controlled by a Master Node known as Nimbus and multiple worker node. A Zookeeper layer in between Nimbus and supervisors of worker nodes is there to provide synchronization functionalities.

A notable property of the physical data model of the real time stream processing architecture is the deliberate placement of the specialized computing element, DeadLineBolt as a computation hungry task into a specific worker process. As the DeadLineBolt is global to the RTopology, a deliberate solo placement perfectly maps the idea as presented in Algorithm 1.

Fig. 3 schematically represents a sample case of mapping of the logical data model of real time stream computation architecture concept to the distributed computing architecture. The Nimbus acts as the master node and controls the entire distributed computation environment. There are only two worker nodes for illustration purpose. Each of them has a supervisor daemon process which maintains the state of the workers with Nimbus through a synchronization layer of Zookeeper. The first worker node has two worker processes whereas the second worker node has only one worker process.

Worker processes present in the first worker node has one Executor in them whereas the worker process of the second worker node has two Executors. Logical computational elements like RBolts, RSpout, DeadLineBolt and PerformanceBolts (designated as PBolts in the diagram) are mapped to the tasks those compose an Executor. It is noteworthy that DeadLineBolt has been placed as a single task into the Executor as discussed earlier

3.4 Time Deadline Management

Time deadline management is the responsibility of DeadLineBolt. In a RTopology, a RSpout has configurable TimeDeadLine and PerformanceGuarantee (with specific default values). Whenever a RSpout fetches tuples from the Streaming Data Source, it forms a TTuple as <TimeDeadLine> & an ExtendedTuple as <tuple, PTuple> (where a PTuple is designated as <PerformanceGuarantee>) and sends them to DeadLineBolt and next RBoltPair respectively.

The DeadLineBolt is synchronized with a Global Clock (with respect to the RTopology) and it records the TimeDeadLine along with the MessageID of the ExtendedTuple. As soon as the RBoltPair receives the ExtendedTuple, it sends a TTupleRequest message (containing the MessageID of the incoming ExtendedTuple to the DeadLineBolt.

Next, DeadLineBolt, matches the MessageID of the TTupleRequest, compares it with the stored MessageIDs and as per the algorithm (currently set as static as mentioned earlier), generates the time deadline

for the individual RBolt (say tRBolt) from the TimeDeadLine and internally computes ReverseTTuple as < tRBolt > to send it back to RBolt.

Upon receiving ReverseTTuple, the RBolt sets the Local Timer (with respect to the RBoltPair) as per <tRBolt> and starts computing the data present in the ExtendedTuple. Once the computation is done, the Local Timer is reset for the next computation.

Thus, the real time stream computation architecture concept maintains a novel and robust, yet simple implementation of time deadline management, which could officially be described through an algorithm as below

Algorithm 1 Time DeadLine Management

Require: $t > 0$ is the default *TimeDead*Line value as preset by *RSpout*
Require: $p > 0$ & $p < 100$ is the default PerformanceGuarantee value as preset by RSpout
Ensure: Message Id of i^{th} message is m_i
Ensure: Message Id from j^{th} *RBoltPair* for i^{th} message is $m_j = m_i$
Ensure: Number of *RBoltPair* is n
Ensure: Dead line time for j^{th} *RBoltPair* is $DeadLine^j$
Ensure: Local timer of j^{th} *RBoltPair* is $lClock^j$

```
TTuple ← t
ExtendedTuple ←<Tuple, p > //Tuple is the part of the original message stream
RBoltPair ← ExtendedTuple
for each i^th message do
   for each j^th RBoltPair of i^th message do
      DeadLineBolt ←< TTuple, m_i>
      TTupleRequest ← m_j
      RBoltPair  sends TTupleRequest to
      DeadLineBolt
      if m_i = m_j then
         DeadLine^j = t/n
         ReverseTTuple← DeadLine^j
         ReverseTTuple is sent back to RBoltPair
         lClock^j ← ReverseTTuple
      end if
   end for
   lClock^j ← 0
end for
```

Above algorithm clearly runs in $O(m * n)$ time for a time slice where m number of messages are being processed with n number of *RBoltPairs*.

3.5 Performance Guarantee Management

As described in the previous subsection, RBoltPair receives an ExtendedTuple which carries Performance Guarantee (in percentage) value in the form of PTuple. PerformanceBolt present within the RBoltPair parses the ExtendedTuple and stores the Performance Guarantee value. However, once RBolt receives the ReverseTTuple from the DeadLineBolt (As mentioned in the previous subsection), starts its Local Timer and starts processing the data. Within the time limit as set by the Local Timer, it keeps on processing the data and at the end of the time limit, RBolt computes the level of performance achieved (say pBolt) and forms an InnerPTuple message as to be sent to corresponding PerformanceBolt. PerformanceBolt receives the InnerPTuple from the attached RBolt and as it already has the Performance Guarantee value with PTuple, it compares if the <pBolt> is greater or equal to Performance Guarantee. If so, it sends a SuccessConfirm message back to the attached RBolt so that the RBolt can start forming the next ExtendedTuple with the data generated through its own processing and a new PTuple value.

In case, the <pBolt> is lesser than the set Performance Guarantee value, then PerformanceBolt would sent a reject notification to the supervisor daemon. It is noteworthy to mention the fact that the result of the last RBolt, as per the RTopology plan, must be stored within a Shadow Result Store temporarily.

It is interesting to discuss, how does a RBolt update PTuple for its next RBoltPair after releasing a SuccessConfirm message for PerformanceBolt associated with it. If $pi-1$ is the performance guarantee value (in percentage) coming through PTuple $i-1$ towards i^{th} RBoltPair and if pai is the performance level achieved (in percentage), then with a constraint $pa_i > p_{i-1}$, the performance guarantee value pi embedded in PTuple$i$ for $(i + 1)^{th}$ RBoltPair could be evaluated as

$$p_i = (p_\theta / pa_i) * 100 \quad (1)$$

where p_θ is the Performance Guarantee value generated by *RSpout* for first RBolt.

It is evident from the above discussion that the cumulative performance level over the entire message pmsg would be identified by the Percentage Level achieved at the last RBoltPair as the InnerTuple which flows from RBolt to PerformanceBolt.

As the supervisor process maintains the state of each worker process and as each worker process maps to a single RTopology, the supervisor process reports pmsg to the synchronization level in parallel. The Zookeeper level collects all the pmsg values corresponding to each RTopology and computes the cumulative performance. If it is acceptable in the background of set Performance Guarantee value, the Shadow Result Store is flushed to the designated data accumulator. All the current discussions can be formalized with a formal algorithm as presented in Algorithm 2 below

Algorithm 2: Performance Guarantee Management

Require: $p_\theta > 0$ & $p_\theta < 100$ is the default PerformanceGuarantee value as preset by RSpout
Ensure: Number of *RBoltPair* is *n* and *RBoltPairs* are identified as $RBoltPair_0$ to $RBoltPair_{n-1}$
Ensure: $RBoltPair_i$ = < *RBolti* , $PerformanceBolt_i$>
Ensure: $RBoltPair_i$ receives $ExtendedTuple_{i-1}$ from $RBoltPair_{i-1}$
Ensure: $ExtendedTuple_i$ = <*Tuplei*, p_i> //Tuple is the part of the original message stream
Ensure: $ExtendedTuple_0$ = < $Tuple_0$, p_θ>
Ensure: p_i is the target Percentage Guarantee which is to be achieved by $RBolt_i$
Ensure: Performance Level achieved by i^{th} *RBolt* i,e pa_i = $InnerTuple_i$ flowing from $RBolt_i$ to $PerformanceBolt_i$
Ensure: $RBoltPair_i$ has a local clock $lClock_i$

TTuple ← t
ExtendedTuple ←< *Tuple, p* > // Tuple is the part of the original message stream
RBoltPair ← *ExtendedTuple*
for each i^{th} *RBoltPair* **do**
    pa_i = Performance Level achieved at
    $RBolt_i$ while $lClock_i$ resets to 0
    if $pa_i \geq p_{i-1}$ **then**
        SuccessConfirm message is sent from $PerformanceBolt_i$ to $RBolt_i$
        pi = pθ pai ∗ 100
    else $pa_i < p_i - 1$
        RejectNotification is sent from $PerformanceBolt_i$ to corresponding supervisor daemon process
    end if
end for
$p_{msg} \leftarrow pa_{n-1}$

Algorithm 2 ensures that a stream message having n RBoltPairs downstream, runs in O(n) time to compute the cumulative performance level which is guaranteed to be greater or equal to the preset value of Minimum Performance Guarantee level, thereby confirming the real time aspect of the stream processing architecture.

In the current section, different fundamental aspects of basic real time stream processing architecture have been discussed. Though this basic architecture is capable of processing streaming messages in real time, there exists enough scope of improvements and optimization for time deadline management. This motivates to extend the current investigations towards forking and queuing theories in subsequent sections anticipating significant improvement in query processing aspects of the real time stream processing architecture.

4. Task Forking Model

In general, the term optimization, as per Oxford Dictionary, refers to “the action of making the best or most effective use of a situation or resource”. Mathematically it relates to maximizing or minimizing some objective function under some constraints, relative to some set, which represents different available choices determining which might be “best” as per the current situation.

In a distributed stream computing environment, optimization can be seen under two paradigms, structural and query processing. Though, structural optimization can be viewed on the background of generic mathematical definition, query optimization often relates to performance gain in terms of lower latency and high throughput.

In the current section, with the term “optimization” discussions have been presented to address how latency time of the proposed system could be reduced maintaining the level of processing throughput intact.

Real time stream computation architecture has to be registered with a Workflow W for any specific business logic. Registration of W essentially means the disintegration of the entire Job of W designated as JW , into a set of n number of tasks (T0 … Tn−1) and to decide how a task Ti would give rise or fork another task Tj where j > i. The registration process gets completed when all the tasks T0 to Tn−1 have been mapped with

corresponding RBolts through API calls. Subsequent subsections discuss about different task forking strategies those could be considered for the real time stream processing architecture and their effect on the real time query processing

4.1 Different Task Forking Strategies

Current subsection describes four task forking strategies namely

- Serial Forking - type f1
- Concurrent Forking - type f2
- Conditional Forking - type f3; and
- Probabilistic Forking - type f4

In serial forking, a task T is decomposed in n smaller tasks $t_0, t_1 \cdots t_{n-1}$ in such way that the t_i finishes first and with the output of t_i , t_{i+1} starts working on and ultimately finishes its job. Hence job of T is considered to be done when all t_0 *to* t_{n-1} have finished their job serially, which essentially means that $T = t_0 + t_1 \cdots + t_{n-1}$
If δ_i is the service time of i^{th} sub task, and if a task T has been decomposed into n subtasks t_0 to t_{n-1}, expression of delay for the serial task T is considered to be

$$\Delta^{T}_{f1} = \delta_0 + \delta_1 + \cdots + \delta_{n-1} = \sum_{i=0}^{n-1} \delta_i \quad (2)$$

In concurrent parallel forking, a task T is decomposed in n smaller tasks $t_0, t_1 \cdots t_{n-1}$ parallel such that job of T is considered to be finished if and only if all the n sub task have finished their job. Hence, if δ_i is the service time of i^{th} parallel concurrent sub task, and if a task T has been decomposed into n sub tasks t_0 to t_{n-1}, expression of delay for the concurrent parallel task T is considered to be

$$\Delta^{T}_{f2} = \max(\delta_0, \delta_1, \ldots, \delta_{n-1}) \quad (3)$$

As all δ_i in Eq. (3) are independent and identical random variables, the Independent and identical distribution of such delays will yield probability of Δ^{T}_{f2} being under a certain value δ is governed by the equation

$$P(\Delta^{T}_{f2} < \delta) = \Pi^{n-1}_{i=0} P(\delta_i < \delta) \quad (4)$$

In conditional parallel forking, a task T is decomposed in n smaller tasks $t_0, t_1 \ldots t_{n-1}$ parallel such that job of T is considered to be finished if any of the n sub task has finished its job. Hence, if δ_i is the service time of i^{th} parallel conditional sub task, and if a task T has been decomposed into n sub tasks t_0 to t_{n-1}, expression of delay for the concurrent parallel task T is considered to be

$$\Delta^{T}_{f3} = \min(\delta_0, \delta_1, \ldots, \delta_{n-1}) \quad (5)$$

Similar to the Eq. (4), the probability of Δ^{T}_{f3} being under a certain value δ is expressed as

$$P(\Delta^{T}_{f2} < \delta) = \Pi^{n-1}_{i=0} P(\delta_i < \delta) \quad (6)$$

In probabilistic forking a task T is forked to only one sub task T_α with a probabilistic value of α where $0 \geq \alpha \leq 1$. But in the background of real time stream processing architecture, where minimum performance guarantee has to be met, probabilistic forking does not find its standpoint, and hence any discussion on this is considered out of scope and hence omitted.

4.2 Optimization of Time Deadline Management with Task Forking Models

Task Forking Models find their application in predicting a preset time deadline which moves through *TTuple* from *RSpout* to *DeadLineBolt* and computing *ReverseTTuple* which flows from *DeadLineBolt* to *RBolts*. The basic real time stream processing architecture, as described in Section 2, only mentions that the time deadline value would be preset by the *RSpout* as a default value. Though, as a primary design criterion, this consideration is not wrong, but this time deadline value has a profound impact on the entire processing throughout *RTopology*. While over estimation of this time deadline value would increase the latency and real-timeness, at the same time it might increase the false positiveness in meeting performance guarantee value. Similarly, at the micro level, if *RBolts* receive a constant and static time deadline value to meet, execution flexibility of the entire architecture would degrade. In this context, subsequent paragraphs discuss how the time deadline management of the architecture could be optimized. Proper estimation of time deadline for the *RTopology* at micro level, denoted as

T^d_M and the time deadline for $RBolt_i$ at micro level, denoted as T^d_{RBolti}, both demand a proper registration of the workflow. This essentially trusts the proper designing of tasks, forking strategies of the tasks and mapping the task functionalities to *RBolts*.

Computation of the delay for any sub graph of *RTopology* can be done recursively by employing Eq. (2), Eq. (3) or Eq. (5) as and when required. A bottom-up design would ultimately compute the delay (time deadline) of the *RTopology* at macro level and at the micro level for *RBolts* stage wise. Algorithm 3 below formally reports the procedures for computing time deadlines

Algorithm 3: Time Dead Line Estimation

Require: *A[n][n]* - Adjacency Matrix for a Workflow represented by a topology having n nodes from v_0 to v_{n-1}

Ensure: Each v_i has an execution length l_i and a delay δ_i
Ensure: (i) *A[i][j]* - 0 denotes no connection between v_i and v_j
(ii) *A[i][j]* - 1 denotes serial forking from v_i and v_j
(iii) *A[i][j]* - 2 denotes concurrent parallel forking from v_i to v_j
(iv)*A[i][j]* - 3 denotes conditional parallel forking from v_i to v_j

select final node v_f such that all $A[f][i] = 0$ where $0 \leq i \leq n-1$
$\delta_f \leftarrow l_f$
select start node v_s such that all $A[i][s] = 0$ where $0 \leq i \leq n-1$
set Time DeadLine $T \leftarrow \delta_{vs}$
select current node $v_c \leftarrow v_f$
build set S and append v_f to S
for each $S_i \in S$ **do**
  while $v_c \neq v_s$ **do**
    append S_i to S where each S_i is a parent of v_c
    for each $v_i \in S$ **do**
      if $A[i][c] == 1$ **then**
        $\delta_i = l_i + \delta_{vc}$
        else $\delta'_i = l_i + max\_v_c$
      end if
    end for
  end while
end for
$\delta_s = \sum_{i=0 \text{ to } m} \delta_i$ such that v_S is the parent of each v_{Si}

Working of the Algorithm 3 can be illustrated with a sample workflow as represented with Fig. 4 below. The diagram shows a sample workflow having thirteen vertices $v_1 \cdots v_{13}$ where each vertex v_i represents a *RBolt* and the streaming message tuples are fed at v_{13} and the processing of tuples end at v_1. For simplicity, only the *RBolts* have been shown in the diagram, leaving other computing elements like *PerformanceBolt* and *DeadLineBolt* as they do not directly contribute to estimation or prediction of deadlines in any of the macro or micro levels.

In the context of Fig. 4 shown below, where the workflow represents a task T, any vertex v_i is considered to have a computing subtask t_i, having its execution length l_i and delay δ_i. Bottom-up approach starts with the final vertex v_1 in this case and assigns l_1 as δ_1. Parent vertices of v_1 are identified as v_2, v_3 & v_4 and as they are all different, three serial forking instances are noted as $v_2 -> v_1$, $v_3 -> v_1$ and $v_4 -> v_1$. Hence delay at v_2 that is $\delta_2 = l_2 + l_1$, similarly $\delta_3 = l_3 + l_1$ and $\delta_4 = l_4 + l_1$

Next, parent of v_2 is computed to be v_5 and v_6 generating a serial forking and hence l_5 and l_6 would be set as $\delta_5 = l_5 + \delta_2 = l_5 + l_2 + l_1$ and $\delta_6 = l_6 + \delta_2 = l_6 + l_2 + l_1$.

Values of δ_7, δ_8 and δ_9 are computed in a same way to be $\delta_7 = l_7 + \delta_3 = l_7 + l_3 + l_1$, $\delta_8 = l_8 + \delta_4 = l_8 + l_4 + l_1$ and $\delta_9 = l_9 + \delta_4 = l_9 + l_4 + l_1$

Now let us suppose that v_{10} forks to v_5 and v_6 concurrently, hence $\delta_{10} = l_{10} = max\ (\delta_5, \delta_6)$. On the other hand, if we consider that v_{11} forks to v_8 and v_9 conditionally, then $\delta_{11} = l_{11} = min\ (\delta_8, \delta_9)$.

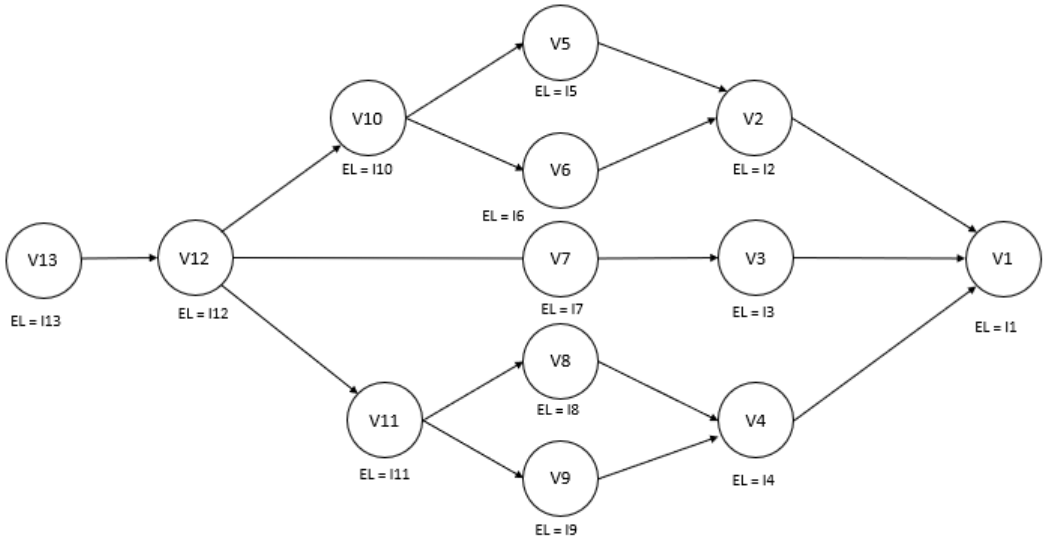


Fig. 4 A Sample Workflow mapped within RBolts

In the diagram we let us further assume that v_{12} forks to v_7, v_{10} and v_{11} concurrently and the source node v_{13} forks to v_{12} serially. The corresponding delays can be computed as $\delta_{12} = \max(\delta_7, \delta_{10}, \delta_{11})$ and $\delta_{13} = l_{13} + \delta_{12}$. Substituting the execution length we get

$\delta_{13} = l_{13} + \delta_{12}$

$\delta_{13} = l_{13} + max\ (\delta_7, \delta_{10}, \delta_{11})$

$\delta_{13} = l_{13} + max\ (\delta_7, (max\ (\delta_5, \delta_6)), min\ (\delta_8, \delta_9))$

$\delta_{13} = l_{13} + max\ ((l_7 + l_3 + l_1), (max\ ((l_5 + l_2 + l_1), (l_6 + l_2 + l_1))), min\ ((l_8 + l_4 + l_1), (l_9 + l_4 + l_1)))$

The optimized delays, thus calculated for the real time stream processing architecture guide the fixation of time deadline management as *TTupe* and *ReverseTTuple*. It is to be noted that the optimization is dependent on the Execution Length of the computing elements or *RBolts*. Hence estimation of the Execution Length has a major impact on proper time deadline management. Different procedures of Execution Length as reported in *[5]* can be employed in this regard. While theoretical processes of Code Analysis, Analytical Benchmarking and Code Profiling estimate a hard bound, practical approach of Past Observation estimates a probabilistic Execution Length. As these discussions are out of context to the present study, in rest of the paper, Execution Length is computed in accordance with measuring the worst-case time complexity against the input size of the computing task presented to *RBolt*. Input size of the task is given by size of the processed data from the previous *RBolt* in *ExtendedTuple* as mentioned in the following equation

$$\text{ExecutionLength} = \text{ExtendedTuple} / \text{PTuple} \qquad (7)$$

In this context, following section studies the proposed architecture in the light of Queuing Model to gain better insights with respect to Transient Solution and Stationary Solution of the system. On the background of real time stream processing architecture discussed so far, while Transient Solution refers to the probability of identifying current *RBoltPair* at specific time synced with the global clock, Stationary Solution describes different performance measures of the system like *RSpout* Stability, Average In-System Concurrent Stream, *RTopology* Engagement Factor etc.

5. Performance Analysis with Queuing Models

As per Kendalls notation, proposed real time stream processing architecture can be described as M/M/1 system, where the distribution of Arrival Time of the stream (*RSpout*'s output) to the *DeadLineBolt* or first *RBoltPair* has a Homogenous Poisson distribution and Service Time or the Delay of the topology, at least theoretically, has an Exponential distribution due to estimated Execution Length having *O(2n)* run time. The number of *RTopology* through which the stream would flow through has been considered by default as *1*.

The generic Transient Solution analysis of the M/M/1 queuing system has a Probability Mass Function $p^k(t)$ which could be expressed in the light of proposed architecture as

$$p^k(t) = e^{-(\lambda+\mu)t}[\rho^{(k-i)/2} . I_{k-i}(\alpha.t) + \rho^{(k-i-1)/2} . I_{k+i+1}(\alpha.t) + (1-\rho). \rho^k \sum_{j=(1+k+2)}^{\infty} \rho^{-jk} I_j(\alpha.t)] \quad (8)$$

where

- pk(t) is the probability of identifying current RBoltPair at specific time t synced with the global clock, Streams arrive at rate λ and 1/μ is mean delay of the RTopology,
- i is the initial number of streams at t = 0,
- ρ = λ/μ,
- α = 2√ λ.μ; and
- Ik is the modified Bessel Function

The probability that the *k* number streams are being processed through *RBoltPairs* is $\pi_k = (1 - \rho).\rho^k$; hence Average In-System Concurrent Stream of the system could be designated by $\frac{\rho}{(1-\rho)}$

Now as per M/M/1, *RTopology* Engagement Factor is the time taken by the stream while it is being place till it was served finally, could be computed as

$$f(t) = \frac{1}{t.\sqrt{\rho}} e^{-(\lambda+\mu)t} \cdot I_1(2t\sqrt{\lambda.\mu}) \quad (9)$$

where I_1 is a modified Bessel function.

Lastly, the response time for streams, as considered as the average time, within which a stream is bound to be served is $\frac{1}{(\mu-\lambda)}$

Modelling the proposed real time stream processing architecture with Kendalls *M/M/1* system offers more realistic and measurable performance metrics as discussed above. The following section would elaborate the experimental procedures to measure the performance and optimization of a sample RTopology that solves a sample use case.

6. Experimental Results

Present section describes the experimental set up and results that further investigates the proposed framework for real time stream computation and it's performance optimization. These experiments generate some valuable insights regarding the theories that have been developed earlier in this paper.

6.1 Experiment Use Case

To experiment with the proposed framework along with its optimization, a specific use case of mobile phone activity in a city has been considered. It analyzes the huge stream containing the Hourly phone calls, SMS and Internet communication of an entire city and decides which is the major activity per day. The result of this analysis when stored in a database, could give an insight of the usage pattern for a mobile activity in a city over a period so that netter business decision could be taken. For this use case, a standard data set [38] has been used from the Kaggle repository.

The experiment is aimed to identify first *n* busiest cellIds in individual field of Calling (considering both call in and call out), SMS (considering both SMS in and SMS out) and Internet Usage. For the sake of simplicity, the experiments omit geographical factors as stored in mi*.csv files and considers only s*.csv files. The experimental data volume is 0.66233 GB spread across 7 files.

6.2 Experimental Set Up

To analyze the huge stream containing the different mobile phone activities of a city, a cluster of 15 nodes had been set up with one nimbus and zookeeper elements. The physical cluster abstracts the *RTopology* having one *RSpout*, one *DeadLineBolt* and three *RBoltPair*s containing three *RBolts* and three *PerformanceBolts* or *PBolts*. All the nodes of the cluster have 10th generation Intel Core I7 CPU @ 2.6GHz processor running operating system Window 10 on 8GB RAM. The application program has been developed using Java in Eclipse 2019-03 IDE. Distributed coordination has been achieved by Apache Zookeeper 3.4.13 installed in one of the designated nodes.

6.3 Objective of the Experiments

The objective of the experiment is not to develop an application with a great business value, rather the objective has been identified as to develop and analyze a sample application with real time data to study

- Whether the proposed architecture offers correct trend of results in real time with minimum performance guarantee (validation)
- Whether the proposed optimization algorithms offer better results
- Whether we receive any insights of the proposed architecture or any clue for further studies

Clearly, the goal of the experimental process is to establish the proposed real time stream processing architecture as a sound framework to analyze any distributed real time streaming data.

6.4 Experiment Methodologies

The experiment uses a Java application to employ and realize a *RTopology* which had been designed to incorporate the use case as described in subsection 5.1. The design of *RTopology* is composed of the following element as shown in Table 1.

As mentioned in the Table 1, There is one *RSpout*, four *RBolts*. each of the *RBolts* coupled with *PerformanceBolt* forms four *RBoltPairs*. There is a single instance of *DeadLineBolt* which provides the global synchronization and time deadline management. In the experimentation process with rudimentary form of the proposed architecture (without any optimization), the hard time deadline for *SplitterBolt* and other *RBolts* have been fixed to be 12 microsecond and 10 microseconds respectively. These deadline values have been averaged out of 1,00,000 executions of the sample computations in non-distributed mode. And the performance level has been set for 95% overall. As the experimental use case has five RBolts, individually the target performance level for the RBolts of the proposed framework have all been set greater than 95% as 97%.

To emulate the streaming computation paradigm, the experimental framework follows the topology diagram as shown in Fig. 5

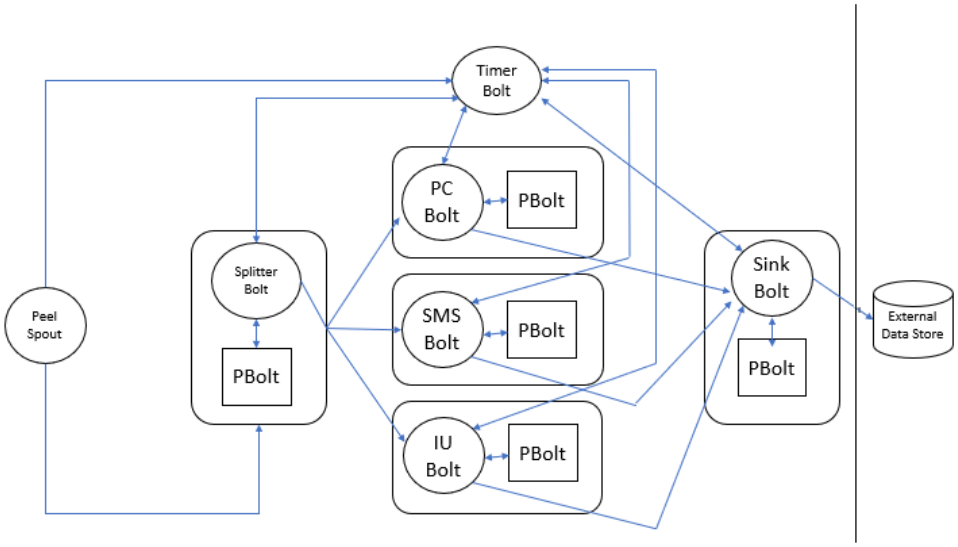


Fig. 5 Topology Under Experiment

6.5 Experiment Results

The first experiment is carried out to identify the benchmark of performance by reading the entire data set and computing 100 busiest CellIds for individual domain type, Phone Call, SMS & Internet Usage and five more queries (designated as q0 to q5 through a Non Distributed Java program running with the same hardware configuration as that of a node as mentioned in Subsection 6.2. The program takes 36.6361481 seconds on an average (out of 300 instances) to compute the result with 100% accuracy. Fig. 6 below shows the distribution of run time in nano seconds over 300 instances of execution of the said program to respond to all the six queries from q_1 to q_6.

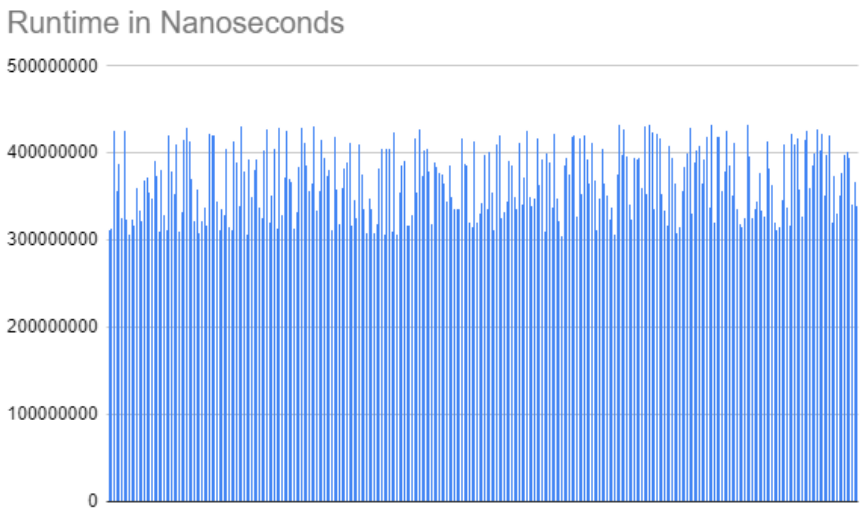


Fig. 6 Temporal Performance of Traditional Program to compute the result of all the queries with 100% accuracy

Fig 7. next shows the radar graph of the query response time for each of q1 to q6 for all 300 instances of execution of the experiment. This diagram conveys the message that the non-distributed and traditional non streaming approach takes time (as all the data points are on periphery of the radar graph) for data preparation and starts responding to the queries late

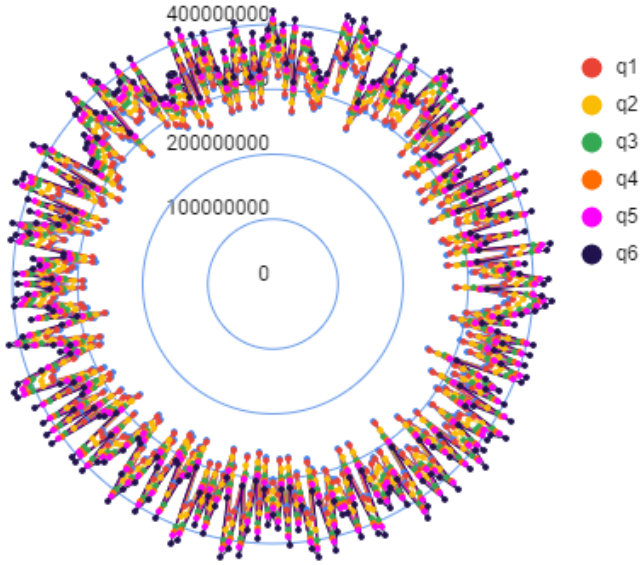


Fig. 7 Query Response Time of the Traditional Program

Second experiment has been executed to judge the merit of the proposed framework without any optimization scheme employed. Performance of each computing elements have been computed by PBolts as per Algorithm 2 with static time deadlines of RSpout and RBolts as 12 and 10 microseconds respectively

On this background, Fig. 8 describes how six queries from q1 to q6 have been responded over time. The proposed streaming processing framework has ensured that the queries start receiving responses much earlier and this experiment has taken 23.0509990 seconds on an average to respond to all the six queries, hence on an average 37.08% gain has been achieved by incorporating distributed and stream computation paradigm.

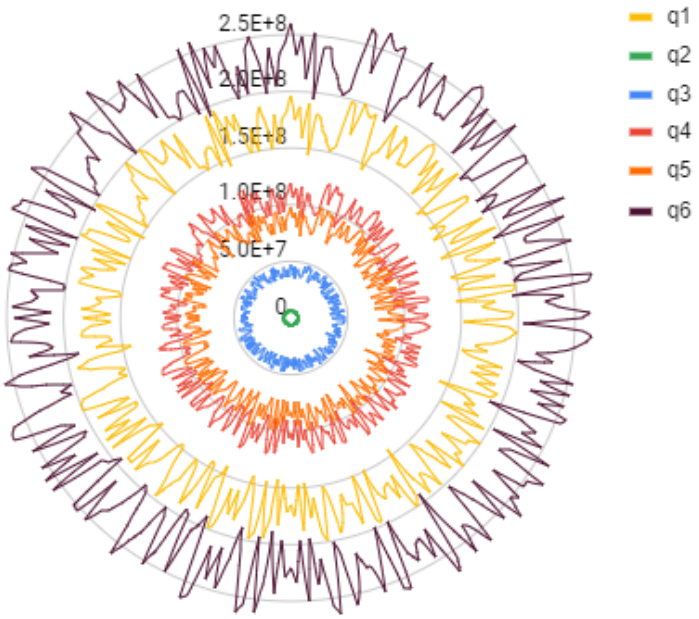


Fig. 8 Performance of proposed framework to respond to each of the six queries

Experiment with non-optimized version of the proposed distributed stream computation ensures minimum performance guarantee to be 95% within a fixed time deadline. Hence, there must be failed cases of data processing in this connection, Fig. 9 shows the average response failure behavior for all the six queries.

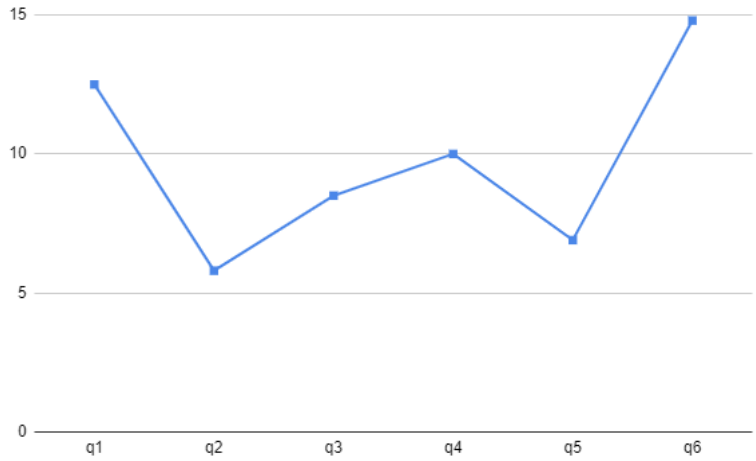


Fig. 9 Failure percentage during data processing

Fig. 10 shows the distribution of response time of the queries in both non-optimized (lighter color) and optimized (dark color) scenario. The temporal gain over different queries have been identified starting from 5 to 15 percent.

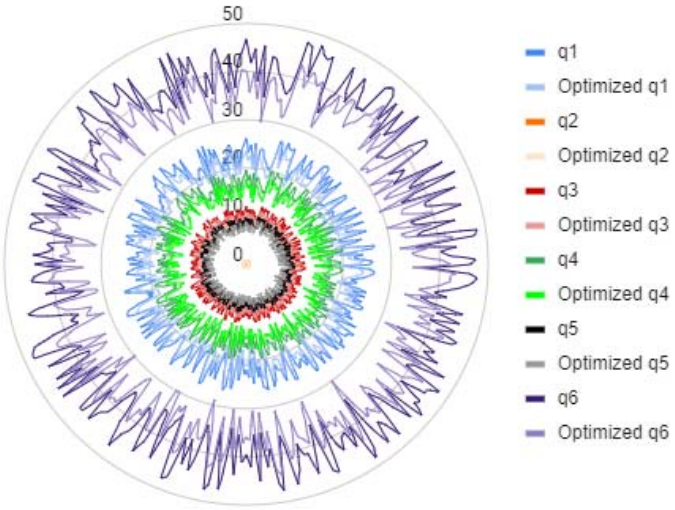


Fig. 10 Query Response Time Distribution under Non-Optimized Vs Optimized Scenario

The temporal gain distribution (As a result of the optimization) for all the six queries has been shown in Fig. 11 below

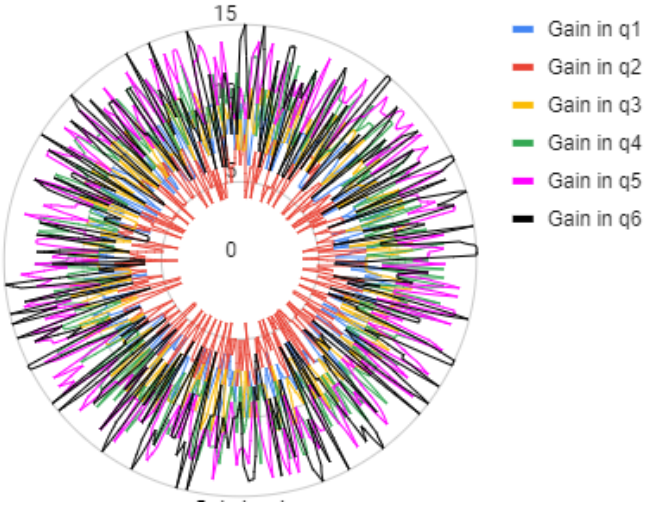


Fig. 11 Temporal Gain Distribution

Similarly, by employing Forking Model based Optimization, some reduction in failure percentage has been observed during query processing. The distribution is shown in Fig. 12 below.

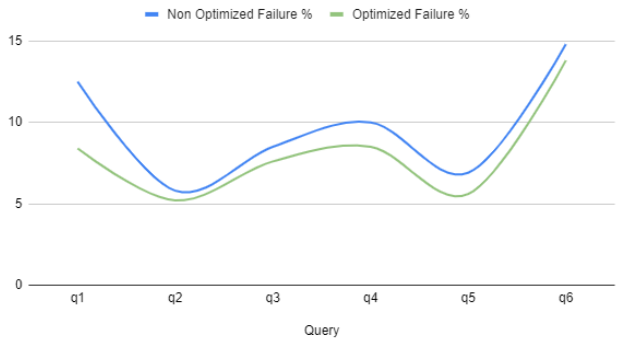


Fig. 12 Reduction of failure percentage during data processing due to forking model-based optimization

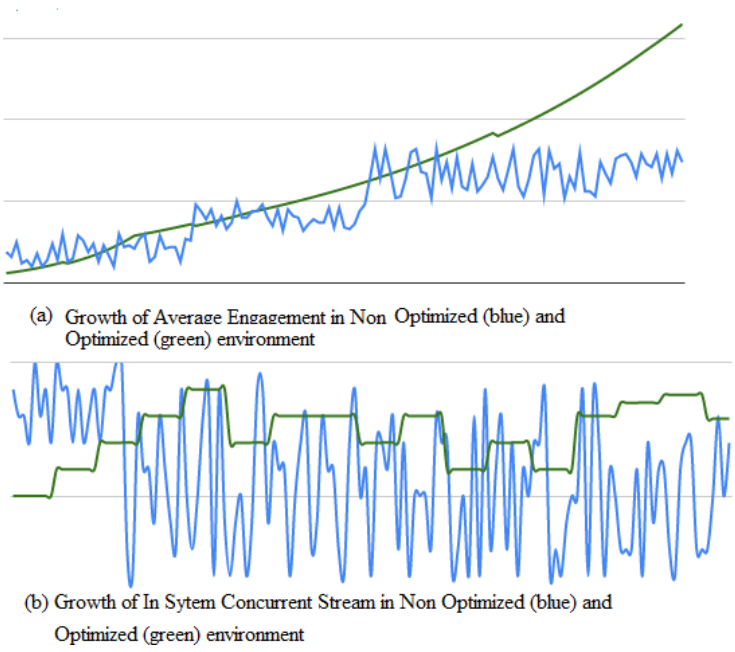
As per the discussions in Section 4, the evident performance metrics of the proposed architecture are (i) Average In-System Concurrent Stream and (ii) RTopology Engagement Factor.

The growth of these parameters over the entire data processing window indicates better stream processing in real time. To compute these two parameters over time, value of the following has been computed

- Message Arrival Rate λ - Based on the fact that the average processing time of PeerSpout has been identified as 12 microseconds, $\lambda = 0.000012$
- Mean Delay $1/\mu$ calculated dynamically by applying $\mu_{SinkBolt} = ExecutionLength_{SinkBolt}$ and tracing back further up to PeelSpout. Eq. (7) is employed to compute ExecutionLength at every RBolt
- Modified Bessel function I1 value is 1.590637 for Ordinal Number 1 and Argument 2

The values as mentioned above are utilized to compute three important parameters as described in Section 4. In the next experiment, growth of these two parameters have been studied both for non-optimized and optimized platform architecture. Fig. 13 (a) and (b) show the average distribution of *RTopology* Engagement Factor and (i) In-System Concurrent Stream respectively for 300 experiment instances over first 800 streams. The results show more stable and steady growth of the parameters in optimized conditions

All the experiment data cited above clearly indicates that the proposed stream processing framework has a potential to process data in distributed real time manner and it's optimization could result in considerable performance upliftment. At the same time, the experiment results also indicate that the nature of query to be processed through the proposed architecture has a critical role in performance evaluation. Moreover, the experiments provide a clear direction on how to measure the performance of proposed real time stream processing platform.



(a) Growth of Average Engagement in Non Optimized (blue) and Optimized (green) environment

(b) Growth of In Sytem Concurrent Stream in Non Optimized (blue) and Optimized (green) environment

Fig. 13 Growth of Average Engagement an In System Concurrent Stream

7. Conclusion

Current paper systematically studies and reports theoretical aspects of real time stream processing architecture capable of processing voluminous data. Logical and Physical data Model of the architecture have been presented in depth. Two polynomial time algorithms have been presented for performance guarantee and time deadline management. Moreover, task Forking based optimization algorithm has been proposed and performance evaluation metrics have been identified through exploring queuing model. A detailed experimental discussion has been presented with visually appealing diagrams. The present study leaves a scope of further research in analyzing the scope and effect of optimizing the computation elements governing the timing tasks and performance management task.

Conflict of Interest

"The authors have no conflict of interest to declare"

References

[1] Zaniolo C, Mining Databases and Data Streams with Query Languages and Rules. Lecture Notes in Computer Science 3933, pp. 1-13, (2005). https://doi.org/10.1007/11733492.2
[2] Kuo Ti, Yang W, Lin K, A class of rate-based real-time scheduling algorithms. IEEE Transactions on Computers, vol. 51, no. 6, pp. 708-720, (2002), https://doi.org/10.1109/TC.2002.1009154.
[3] Li Peng, Ravindran B, Fast, best-effort real-time scheduling algorithms. IEEE Transactions on Computers, vol. 53. pp. 1159 - 1175, (2004), https://doi.org/10.1109/TC.2004.61.
[4] Buttazzo, Giogio C, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications", 3rd Edition. Book Published By: Springer Publishing Company, Incorporated, ISBN: 978-1-4614-0675- 4
[5] Shin K G, Ramanathan, P, Real-time computing: a new discipline of computer science and engineering. Proceedings of the IEEE, vol. 82, no. 1, pp. 6-24, Jan. (1994), https://doi.org/10.1109/5.259423.
[6] Wang Q, Wang H, Jin H, Dai G, Design and evaluation of priority table based real-time scheduling algorithms. IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, (2003). Volume: 2, Pages: 1294 - 1299 vol.2, https://doi.org/10.1109/RISSP.2003.1285779
[7] Xiangbin Z, Shiliang T, An improved dynamic scheduling algorithm for multiprocessor real-time systems. Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, (2003), Pages: 710 - 714, 10.1109/PDCAT.2003.1236397
[8] Zhang L, Huang J, Zheng Y, Scheduling algorithms for parallel real time systems. 1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM. 10 Years Networking the Pacific Rim, 1987-1997, (1997), Volume: 2, Pages: 968 - 971 vol.2, 10.1109/PACRIM.1997.620421
[9] Zhang L, Huang J, Zheng Y, Scheduling algorithms for multiprocessor real-time systems. Proceedings of ICICS, 1997 International Conference on Information, Communications and Signal Processing (1997), Volume: 3, Pages: 1470 - 1474, https://doi.org/10.1109/ICICS.1997.652236
[10] Kim K H, Fundamental research challenges in real-time distributed computing. Proceedings. 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, (2004). FTDCS 2004., 2004, pp. 2-9, https://doi.org/10.1109/FTDCS.2004.1316586.
[11] Goodloe A E, Monitoring distributed real-time systems: a survey and future directions. Book Published By: National Aeronautics and Space Administration, Langley Research Center (2010)
[12] M¨alardalen N P, Distributed Real-Time Systems Survey - Scheduling and Communication. (2009)
[13] Muthukrishnan S, Data Streams: Algorithms and Applications, Foundations and Trends® in Theoretical Computer Science: Vol. 1: No. 2, pp 117-236. https://doi.org/10.1561/0400000002
[14] Aggarwal C, Data Streams: Models and Algorithms. https://doi.org/10.1007/978-0-387-47534-9.
[15] Babcock B, Babu S, Datar M, Motowani R, Widom J, Models, and Issues in Data Stream Systems. Proceedings of the ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems, pp. 1–16 (2002) https://doi.org/10.1145/543613.543615
[16] Minos G, Johannes G, Rastogi R, Querying and mining data streams: you only get one look a tutorial. SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, (2002), 635. https://doi.org/10.1145/564691.564794.
[17] Arasu A, Babcock B, Babu S, Cieslewicz J, Datar M, Ito K, Motwani R, Srivastava U, Widom J,STREAM: The Stanford Data Stream Management System, Technical report, Stanford InfoLab, (2004)
[18] Klein C, Rumpe B, Broy M, A stream-based mathematical model for distributed information processing systems. SysLab system model. ArXiv e-prints, September 2014. arXiv:1409.7236
[19] Yamamoto S, Introduction to Mathematical Modeling and Computation." Online Lecture Notes, Laboratory of Mathematical Modelling and Computation, Dept. of Computer and Mathematical Sciences, Tohoku University, (2018).
[20] Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel J M, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, Bhagat B, Storm@Twitter, SIGMOD '14: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, (2014), pp 147–156, https://doi.org/10.1145/2588555.2595641
[21] Puttaswamy R, Scale-Out Algorithm For Apache Storm In SaaS Environment, University of Nebraska (2018).
[22] Jose R, Jos´e M, Simona B, Diego M, Giorgos G, Vasilis P, Quantitative Analysis of Apache Storm Applications: The NewsAsset Case Study. Information Systems Frontiers. 21. https://doi.org/10.1007/s10796-018- 9851.
[23] Nivash J, Ebin R, Dhinesh Babu L D, Nirmala M, Kumar M, Analysis on enhancing storm to efficiently process big data in real time. (2014) pp1-5. https://doi.org/10.1109/ICCCNT.2014.7093076.
[24] Yang W, Liu X, Zhang L, Yang L T, Big Data Real-Time Processing Based on Storm, 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, (2013), pp. 1784-1787, https://doi.org/10.1109/TrustCom.2013.247.
[25] Chardonnens T. Cudre-Mauroux P, Grund M, Perroud B, Big data analytics on high Velocity streams: A case study. IEEE International Conference on Big Data, (2013), pp. 784-787, https://doi.org/10.1109/BigData.2013.6691653.
[26] Van T, Chuan-Ming L, Goodwill N, Big data stream computing in healthcare real-time analytics. (2016) pp 37-42. https://doi.org/10.1109/ICCCBDA.2016.7529531.
[27] Dawei S, Rui H, A Stable Online Scheduling Strategy for Real-Time Stream Computing Over Fluctuating Big Data Streams. IEEE Access. PP. 1-1. https://doi.org/10.1109/ACCESS.2016.2634557.
[28] Matsuura H, Ganse M, Suzumura T, A Highly Efficient Consolidated Platform for Stream Computing and Hadoop, IPDPSW '12: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, (2012) pp 2026–2034, https://doi.org/10.1109/IPDPSW.2012.252
[29] Barbieru C, Pop F, Soft Real-Time Hadoop Scheduler for Big Data Processing in Smart Cities, IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), (2016), pp. 863-870, https://doi.org/10.1109/AINA.2016.122.
[30] Borthakur D, Gray J, Sen Sarma J, Muthukkaruppan K, Spiegelberg., Kuang H, Ranganathan K, Molkov D, Menon A, Rash S, Schmidt R, Aiyer A, Apache hadoop goes realtime at Facebook, SIGMOD '11: Proceedings of the ACM SIGMOD International Conference on Management of data, (2011) Pages 1071–1080, https://doi.org/10.1145/1989323.1989438
[31] Thammawichai M, Kerrigan E C. Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors. Real-Time Syst 54, 132–165 (2018). https://doi.org/10.1007/s11241-017-9291-6
[32] Draskovic S, Ahmed R, Huang P et al. Schedulability of probabilistic mixed-criticality systems. Real-Time Syst 57, 397–442 (2021). https://doi.org/10.1007/s11241-021-09365-4

[33] Agrawal, K., Baruah, S., Ekberg, P. et al. Optimal scheduling of measurement-based parallel real-time tasks. Real-Time Syst 56, 247–253 (2020). https://doi.org/10.1007/s11241-020-09346-z
[34] Ozbayoglu M, Kucukayan S, Dogdu E, A real-time autonomous highway accident detection model based on big data processing and computational intelligence. 2016 IEEE International Conference on Big Data (Big Data) (2016), pp. 1807-1813, https://doi.org/10.1109/BigData.2016.7840798.
[35] Munshi S, Saha S, Goswami R T, Real Time Big Data Processing: A Comparative Study of Existing Approaches". IEEE 16th India Council International Conference (INDICON)(2019). https://doi.org/10.1109/INDICON47234.2019.9029096
[36] Munshi S, Saha S, Goswami R T, "Extending Storm API for Real Time Computing on Big Data". IEEE 16th India Council International Conference (INDICON)(2019). https://doi.org/10.1109/INDICON47234.2019.9028971
[37] Kulkarni S, Bhagat N, Fu M, Kedigehalli V, Kellogg C, Mittal S, Patel J.M, Ramasamy K, Taneja S, "Twitter Heron: Stream Processing at Scale" SIGMOD '15: Proceedings of the 2015 ACM SIGMOD International Conference on Management of DataMay 2015 Pages 239–250, https://doi.org/10.1145/2723372.2742788
[38] https://www.kaggle.com/datasets/marcodena/mobile-phone-activity

Authors Profile

Shiladitya Munshi is a doctoral student of Department of Computer Science & Engineering, The Assam Kaziranga University. He works at the capacity of Assistance Professor at Department of Information Technology in Techno International Newtown. Shiladitya considers Real Time Big Data Analytics as his research area.

Prof Sajal Saha has a wide experience as an engineering faculty, researcher and academic administrator. Currently Prof. Saha is heading Meghnad Saha Institute of Technology as Principal. His research domain includes Distributed Computing, Big Data and Psycho Analysis.

Prof R T Goswami has a vast experience in teaching, research and academic administration . Currently Prof. Goswami is acting as Director at Techno International Newtown. His research domain includes Database, Distributed Computing, Computer Networks, Big Data and Data Analytics.