The thesis examines how to optimize the performance of data stream processing systems (DSPs) on multicore architectures. The topic under study is timely and has important practical applications. The thesis consists of three contributions. The first contribution (Chapter 3) presents a study and evaluation of the scalability of two DSPs (Apache Storm and Flink) and proposed two optimization techniques (non-blocking tuple batching and NUMA-aware executor placement) to address the performance issues faced by existing DSP designs. The second contribution (Chapter 4) presents a DSP named BriskStream which is based on a novel NUMA-aware streaming execution plan optimization technique to achieve higher throughput and lower latency. The third contribution (Chapter 5) presents a DSP named TStream for scaling transactional state management in data streaming applications.

In my opinion, the candidate has made significant research contributions on the topic of optimizing data stream processing systems. The proposed techniques are non-trivial and novel, and each of them has been experimentally demonstrated to be effective with an implemented system. The results of the first two contributions have been published in ICDE 2017 and SIGMOD 2019 conference papers, and the result of the third contribution has been submitted for publication. I therefore recommend that the candidate be awarded the Ph.D. degree for his research contributions.

1 Detailed Comments

1. Page 13: The example given in the first sentence in Section 2.4.2 does not seem to be correct. Please check its correctness and revise if necessary.

2. BriskStream introduces three heuristics to reduce the optimization search space. It would be interesting to compare the relative effectiveness of these heuristics in the experimental evaluation.

3. Page 88: the discussion on event timestamps references the motivation application in Section 4.2, but the referenced example is not related to the event timestamp issue.

4. In terms of the thesis presentation, the three pieces of work contained in the thesis are presented as three independent papers. I would suggest that the thesis be revised to better integrate the background material into perhaps a chapter on its own. Specifically, the background on multi-socket multi-core processors and data stream processing systems in Section 3.2 (Preliminaries and Background) and Section 4.2 (Background) could be covered in a separate chapter before the three main chapters. Similarly, the diagrams in Figures 3.5 and 4.7 for the same application experimental setup could also be combined to avoid repetition. The usage of "in this paper" (pages 16, 45, 65, and 119) should be replaced with "in this thesis". The usage of "due to space limitation" (pages 84 and 91) is not appropriate for a thesis and should be revised accordingly.

5. In Chapter 5, several references to sub-figures would be clearer by referring to the sub-figure labels instead of using descriptive phrases (e.g., "bottom left of Figure 5.3").

6. There are several incomplete/inconsistent entries in the bibliography (e.g., [Mea] has missing conference year, some conference publication entries include venue information while others do not).

7. The following are minor typographical errors that should be rectified in the thesis:

   (a) Page 3: "Apache Strom"

   (b) Page 12: "th shared"

   (c) Page 88: "By taking toll query processing"

Section B: Examiner's Detailed Comments

Thesis: **Scaling Data Stream Processing on Multicore Architectures**
Candidate: Shuhao Zhang

This thesis discusses scaling data stream processing (DSP) on shared-memory multicore architecture. The main aim is to support the continuously processing of data streams efficiently in real-time.

The thesis focuses on DSP application on shared-memory platforms and made two key contributions. A new stream optimized execution plan called relative-location aware scheduling that is aim to scale to hundreds of NUMA core but supported by results on more modest 8-socket machines. A new stream transactional state management that decoupled state management from streaming computational logic to address the high cost of synchronization shows a 6.8 times throughput improvement on a 40-core system. I believe these new insights and approaches advance our knowledge on DSP. With the proliferation of IoT devices (and thus data streams), this is an important area of growing interest. The thesis is well written and organized and succinctly captures current state-of-the-art in this area with fair comments and assessment of the topic. I note that the thesis work has been published in two top places and the candidate is also a co-author of another 5 published papers.

In my view, the thesis meets the requirements expected of a good thesis and I have no hesitation to **recommend the acceptance of the thesis**.

I hope the following comments and clarifications will help to further improve the clarity and presentation of this thesis.

1. General Comments

   a. **DSP application**: To have a better understanding of the problem and how the proposed approaches addressed and to what extend it addressed the problem, it is useful to characterize the expected data size and the computing demand of this problem among others. What is the total expected incoming data stream (arrival) rate, computation demand to process data and amount of data movement to support processing, computation-to-communication ratio (compute-intensive, data-intensive where memory and I/O do most of the work or both)? In particular, data rate will only grow and is likely to grow at a much faster rate than improvements in processing. How would this impact DSP scalability on shared-memory systems?

   b. **Shared-memory multicore**: I am ok in pushing the limit of the problem on shared-memory systems versus "cluster of low-end servers". But it is well-known that shared-memory has much lower scalability limit – perhaps this should be highlighted. Next, 6.8x throughput improvement is impressive but how far are we from optimal or what is the speedup achieved on the 40-core system?

c. **Performance proxy**: throughput versus time – The evaluation is biased towards using *throughput* as the performance proxy (though I see time reported in the thesis but not evaluated much further). Throughput is key in batch processing and from a system's perspective but as I understand, this thesis focuses on real-time stream processing, i.e., continuously process streamed data and also presumably dished out data analytics on-the-fly too. I would argue that *time*, i.e., low latency is a key focus in stream processing and what users care for. Next, time and latency performance diverge and you can tradeoff one for the other (as you rightly pointed out in page 9, page 15 – "support very low latency processing, which is one of the key requirements in many real applications"), i.e., high throughput can be achieved but at the expense of high latency. To what extend is 6.8 times improved in throughput achieved (or to what extend) at the expense of latency if any? Consider putting in some comments to provide a more balanced view.

2. Detailed Comments
   a. Chapter 2 – I enjoyed reading this chapter, it is short and clearly captured the key issues and state-of-the-art. A better organization is 2.1 … hardware, 2.2 DSP Processing (with current 2.3 and 2.4 as subsections), 2.3 Performance…. This tells the readers what are current platforms, what are the key issues in DSP processing and how you evaluate the performance.

   b. Chapter 3 – This chapter contains a wealth of data and insights in evaluating 3 common design aspects using 2 DSP systems.

      In case I miss out, is there any different between "pipeline processing" in the thesis and pipeline of jobs in Spark (Streaming)?

      Next, it is quite well known that in-memory computation, together with support for iterative computations and generic operators can improve performance by several orders of magnitude compared to a pipeline of MapReduce jobs. In comparison with the 3 aspects selected, is in-memory computation a technique of much lesser important in improving stream performance?

      Figure 3.7 – what is the rationale/objective behind the sub-classification of time? Front-end stalls and back-end stalls are undefined – not sure what constitute these and thus unsure how to interpret the results? By forward reading, I note that front-end stalls can be inferred from 3.5.2 but term used must be defined before it is use. What about back-end stalls – not discussed further?

   c. Chapter 4 – This chapter is well organized and written but there are some gaps. The main objective is to find a good stream execution plan. This thesis proposed RLAS, largely based on a branch and bound approach and went on to reduce the solution space using 3 heuristics.

      Section 4.1 – Candidate articulate that streaming processing on multi-core is challenging, i.e., "varying processing capability and resource demand of each operator

due to varying remote memory access penalty under different execution plans". Candidate also commented that current DSPS "are mainly designed and optimized for scaling-out using a cluster of low-end servers", so why bother about making DSP more efficient on multi-core beyond just being challenging?

Despite reading chapter 4 that shows BriskStream is better than Storm and Flink, I remain to be convinced that given the same commodity server hardware and a shared-memory servers with roughly the same compute/memory/network performance/capacity and cost, shared-memory can outperform distributed-memory. Note that a NUMA architecture is a distributed system (low-end cluster servers) with better network latency – but this performance edge is likely to be nullified because of the more complex hybrid shared- and distribute-memory architecture in a multi-socket multi-core systems and the overhead incurred in having a more complex execution plan.

In-fact, the crux of the problem and the interest in advancing shared-memory system performance is not the above but that while cluster servers are scalable, each server node still adopt shared-memory and thus it is impt to optimize single shared-memory node performance. You could highlight this to make a stronger case for your work.

Comparing BriskStream with Storm and Flink – Add some justifications that the comparison is fair since these are designed with different memory models and at different levels of maturity. Figure 4.8, the bar chart uses different base/denominator so may be misleading to compare speedup between Storm and Flink? Improvement over Storm and Flink are good but any way to work out (mathematically or using the model) or infer the BriskStream optimal performance? [perhaps something similar to chapter 5, figure 5.1.a is good – shows the large gap.]

4.6.3 "Assume input ingestion rate (I) is sufficiently large and keeps the system busy" – this assume is sound, explain how this is determine/set in the experience (sorry I may have missed this). Beyond "I", what is the assumption on data streams, i.e., bounded or unbounded or does it matters? There are some work on unbounded data stream that requires a difference processing approach. Put some comments to tighten your input assumption.

d. Chapter 5 – This chapter treats transactional state management with a good degree of depth and objective is "focuses on achieving a reasonable latency level, with high throughput". The primary analysis appears to be on throughput but with runtime being secondary (figure 5.1 and 90% latency in 5.13b and 5.14). I think there is enough result to have a stronger justification of your objective statement in particular in the conclusion (currently it is more of a summary without quantifying key results). For examples, highlight improvement in throughput and qualify the "reasonable end-to-end latency".

23 Aug 2019 - Teo Yong Meng

Student Name : ZHANG SHUHAO
Comments from E3

Comments on “Scaling data stream processing on multicore architecture”

Summary

This thesis addresses a key question, which has been largely overlooked by prior researchers: how to boost performance/efficiency of stream analytics on individual machines. As the thesis rightly points out, mainstream analytics engines today often focus on “scaling out”. This is because typically they are designed for data centers under the assumption that there will be enough idle machines lying around.

Following a “measure then build” approach, this thesis studies three aspects of single-node stream processing engines:

1. Profiling popular engines, especially for understanding the microarchitecture implications. This study uncovers significant inefficiency, motivating the subsequent system design efforts.
2. Designing BriskStream, a stream analytics engine for multicore/manycore machines. The key idea behind BriskStream is to formulate the problem of operator placement over NUMA cores and automate the process at run time.
3. Designing TStream, which further augments BriskStream to support scalable transactional statement management.

Overall, the thesis contains original contributions, is well written, and meets the bar of a Ph.D. thesis in my opinion.

Next, I describe a few additional items that the author may consider discussing in the thesis. The author can adopt as he sees fit.

Impact of platforms

The proposed engines seem to be evaluated on brawny machines each with 4-8 sockets. Such machines are important platforms; they, however, incur much higher per-core cost as compared to commodity servers (1 or 2 sockets) widely used in data centers today. The author may want to discuss the applicability of BriskStream/TStream designs on commodity servers. On one hand, do we expect to see BriskStream/TStream showing similar or lower benefit on these machines? On the hand, with the improved performance on scale-up servers, can BriskStream/TStream provide better performance/$ on scale-up servers than running today's engines (e.g. Flink) on commodity servers?

Ingestion design

From the thesis writing, I could not find a description of ingestion – how data flows into the engine (do I miss anything)? Ingestion is an essential component of the proposed engines. It is likely of particular interest when the engine is operating with so many cores.

I understand that much prior work on stream analytics omitted ingestion design. Nevertheless, I hope that the author could discuss such an issue and offer thoughts on ingestion design, e.g. to cater to NUMA effect and a large core count.

Motivate the need for high performance

An underlying assumption is that we do need higher performance for stream analytics. Why today's stream analytics performance is still unsatisfactory? The thesis could be made stronger by identifying and describing a few scenarios where higher throughput, lower latency, and/or higher-efficiency are desired.

The implications on edge processing

By targeting scale-up servers, the thesis seems to assume a cloud environment. As IoT emerges and much computation is moving closer to data sources, can the proposed techniques benefit stream processing at the edge too? For instance, the notion of transaction processing would still be valid on the edge; the need for throughput may be lower; the need for latency may be stricter. It would be interesting to see related discussion.

Literature

I liked the Literature review section and would like it to be a "must read" for whoever will work on single-node stream analytics. The author may want to consider the following revision.

1. Discuss the **driving applications** of stream analytics. It is great to discuss how engine designs evolved over time (page 5—7), which I think is fundamentally driven by the change of applications over time. For instance, StreamIt was mainly for supporting signal processing and media workloads while Storm/Flink/Trill/etc were for analyzing server logs, user click streams, etc. This helps clarify "why the proposed multicore/NUMA optimizations were not done before"?
2. Add some discussion on stream model. I think the thesis (implicitly) assumes in-order event arrival which is fine. Yet, out-of-order streaming has received much attention and may be worth discussion.
3. Reference entries shall be complete and in a consistent format. Example entries with glitches include Mea and Seaa.