最初は他の論文でも紹介しようと思ってて....
The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors (SOSP '13)

結局表題のこちらにしました
https://www.usenix.org/conference/atc14/technical-sessions/presentation/hu

この文書の諸注意

柏原はストリーム処理系の専門家ではない。
説明の不足やミスがあってもご了承いただきたい。

あらすじ

読むモチベーション

ストリーム処理系というのと、decentralized, DHTなどのキーワードに惹かれて。

この研究のチャレンジ

jobごとにdecenteralizedな “many masters”アーキテクチャの構成でスケールさせる
メモリ効率の良い compressed buffer tree (CBT)にtemporaryデータを格納する
worker processでの集約（aggregation）にはshared reducer trees(SRT)というテクニックが使われる

論文の結論

- good performance and scalability
- future work: explore task isolation

イントロ 背景+ストーリー

ストリーム処理が流行っている
ビジネスマーケティングでの意思決定
SNSにおけるスパム検知
などがストリーム処理でやりたいこと。

具体的な例として(product, click) pair の列からtop-k求めたい、とか

イントロ その2：ストリーム処理系で求められるもの

- flexibility

既存のシステムには柔軟さがない
Spark Streamingなどはある種のバッチ計算の連続を扱う
Stormだとデータフローのグラフ（頂点）ごとに非同期処理を行ったりする
これらはcomposableでないらしい

私の観点からすると、アプリケーションの開発者はストリーム処理系とそのAPIにあわせてシステムを構築するものだと思っていたが、著者の主張ではストリーム処理系自体にもうちょっと柔軟性を持たせたい模様。

- scaling with job diversity
  MapReduceだとsingle master/-many workers スタイル。centralized。中央集権型。
  中央集権型だと、複雑なpipeline/cyclic dataflowだとボトルネックになりがちだよねー

- obtaining high performance for incremental updates
  既存のシステムでは困難らしい
  インメモリでのhashtableのようなデータ構造では、過去の状態と新しい状態を持つのが...
    TODO: 具体的にどんなデータ構造がよくわからん

- ELF: Efficient, Lightweight, Flexible stream processing system
  FがFastからFlexibleに変わった？
  figure 2
  DHTのPastryを使う。

Design

Overview

Figure 3

level1: webserver + Agent
level2: master and set of workers (ここでDHTのオーバーレイネットワーク構築してる？)
level3: ELF API, プログラマのbatch/streaming analysis, interactive queries向け

CBT-based Agent

DHT-based SRT

Implemantation

System Architecture

Figure 8

Agentの役割

Job master

- Job builder:
- Job tracker:
- Job feedback:
- Job coordinator:

Job worker

- Input receiver:
- Job parser:
- PAOs execution:

Consistency

複数のノードをまたいで計算をしたとき、一貫性の問題がある。どうやって解決しているか。
他のシステムだと、Borealisはノード間の同期で、Stormは一貫性無視らしいが。。？

CBT + timestampで担保しているような記述に見えるが。

Fault Recovery

ELFは次の2つの障害に対応している。

- transmission faults
- agent failures

transmissionについてはXOR protocolで検出、recordのintegrityチェックして、エラーがあれば再送する。
XOR protocolとは何なのか、実装は読んでいてよくわからない。XORで通信データなどのチェックサムを取っているのかもしれない。

agent failuresについては、CBTとP2Pで対処。障害が発生するとキャッシュしていたCBTからSRTが再構築される。すべてのPAOはチェックポイントから新しく再計算される。
翻訳したが、よくわからん。

Straggler Mitigation

1st “mirroring straggler” 遅いと推測されるタスクのコピーを近傍のagentで実行
2nd “leaping straggler” 遅延のあるスナップショットの計算はスキップし、次の期間になったらストリームの計算を再開する

論文の見所

decentralized (中央集権でない) architecture for scalable streaming procesing

論文に関するツッコミ所（質問したい、確認したい所）

- どうやってdecentralizedを実現しているのか？
    - CBT based agentに書いてあるように
- ジョブ（タスク）はどうやって管理してるのか？
    - Job master(agent)が管理していると思われる
- 本当にスケールするのか？
    - DHTなのでしそうな気はする
    - DC内で1280台で実験している
- 実装は？
    - OSSではない
    - 研究のためのプロトタイプ

Keywords

CBT: Compresssed Buffer Tree

http://www.pdl.cmu.edu/PDL-FTP/HECStorage/git-cercs-12-08.pdf
hash-based aggregator
buffer: key/pair単体で圧縮するのは計算/空間効率が悪いので、バッファ単位で圧縮する
CBTの中でPAOを扱う

PAO: Partial Aggregation Object

CBTの論文中で定義されている。

SRT: Shared Reducer Tree

総評

2014年の資料ということもあって、比較的新しめの論文（GoogleのMilwheelなど）や処理系（Spark Streaming等）にもRelated Workなどで言及できている様子はある。他システムと比べた評価実験もやっている。

インメモリのデータ構造であるCBTと、その中で扱っているPAOの概念については、CBTの論文を読まないとほとんど中身がわからない（読めていない）。

DHTのアルゴリズムになぜPastryを使ったのかは書いていない。しかし、many-mastersと呼ぶアーキテクチャでDHTを使ったストリーム処理系を構築したという点では独創的である。

動画のメモ

6:25 既存手法、1 master, multi workers
7:21 prorpose: DHT based
8:36 DHT hashingの説明
9:36 many masters, many workers
9:54 software architecture
st layer: located in each Web server, DHT
2nd layer: key-value pairs stored.
3rd layer: Open to programmer. Programmer can implement diffrent job/data flow and so on.
10:49 ELF Design - ELF have ELF QL
example: micro promotion. product and clicks key/value pairs.
Compressed Buffer Tree: (1) insert, (2) flush, (3) empty actions
stateful, asynchronous, and synchronous execution
12:48 ELF Design - SRT (Shared reduced tree)
よくわからん
15:05 Pipeline, cyclic dataflow, and cross-job coordination
15:20 Using Elfexample
15:30 Evaluation
- latency
related work: Storm and Muppet
17:36 ZConclusion
- good performance and scalability
- future work: explore task isolation

Muppet: MapReduce-Style Processing of Fast Data
http://vldb.org/pvldb/vol5/p1814_wanglam_vldb2012.pdf