Big Data Systems: Achieving Performance and Usability

Peter Pietzuch
prp@imperial.ac.uk

Large-Scale Distributed Systems Group
Department of Computing, Imperial College London
http://lsds.doc.ic.ac.uk

Big Data Excellence, Berlin, Germany – March 2017
Machine Learning For Web Data

- How to maximise user experience with relevant content?
- How to analyse “click paths” to trace most common user routes?

Example: **Online predictions for adverts** to serve on search engines

Solution: **AdPredictor**
- Bayesian learning algorithm ranks ads according to click probabilities

\[ y \in \{-1, 1\} \]

\[ \text{predict} \quad \text{update} \]
Data Mining of Social Networks

Twitter Cascade Detection

Peter Pietzuch - Imperial College London
Intelligent Urban Transport

Instrumentation of urban transport

- Induction loops to measure traffic flow
- Video surveillance of hot spots
- Sensors in public transport

Potential queries

- How to detect traffic congestion and road closures?
- How to explain the cause of congestion (public event, emergency)?
- How to react accordingly (e.g., by adapting traffic light schedules)?
Big Data Applications Enabled by Systems

Data centres allow processing at scale
- Commodity servers
- Fast network fabrics
Task vs Data Parallelism

Big Data

n machines in data centre

Results

Task parallelism:
Multiple queries

Data parallelism:
Single query

select distinct W.cid
From Payments
partition - by 1 row
as W,
Payments
range 300 seconds
where W.cid = L.cid
and W.region != L.region

select distinct W.cid
From Payments
range 300 seconds
as W,
Payments
partition - by 1 row
as L
where W.cid = L.cid
and W.region != L.region

select highway, segment, direction, AVG(speed)
from Vehicles
range 5 seconds slide 1 second
group by highway, segment, direction
having avg < 40

Peter Pietzuch - Imperial College London
Idea: Execute multiple data-parallel tasks on cluster nodes

Tasks organised as dataflow graph

Almost all big data systems do this: Apache Hadoop, Apache Spark, Apache Storm, Apache Flink, Google Dataflow, Google TensorFlow, ...
Tension between performance and algorithmic expressiveness
1. High Performance with New Hardware

High-throughput processing

- **Facebook Insights**: Aggregates 9 GB/s, < 10 sec latency
- **Feedzai**: 40K credit card transactions/s, < 25 ms latency
- **Google Zeitgeist**: 40K user queries/s (1 sec windows), < 1 ms latency
- **NovaSparks**: 150M trade options/s, < 1 ms latency

Low-latency results

Peter Pietzuch - Imperial College London
2. Expressing Machine Learning Algorithms

- Online machine learning, data mining
- Increasing complexity

Data filtering ➔ Complex pattern matching ➔ Data queries

T1(a, b, c)  
T2(c, d, e)  
T3(g, i, h)
Servers have many **parallel** CPU cores

Servers with **GPUs** common
  - GPU have even more specialised cores

New types of **compute accelerators**
  - Xeon Phi, Google's TPUs, FPGAs, ...

*Peter Pietzuch - Imperial College London*
SQL provides well-defined declarative semantics for queries
- Based on relational algebra (select, project, join, …)

Example: Identify slow moving traffic on highway
- Find highway segments with average speed below 40 km/h

```
select highway, segment, direction, AVG(speed) as avg
from Vehicles[range 5 sec slide 1 sec]
group by highway, segment, direction
having avg < 40
```
Queries must operate over **finite** amounts of data at a time.

Processing Big Data with Windows

Peter Pietzuch - Imperial College London
Naive strategy parallelises computation along \textit{window} boundaries

\textbf{Window-based} parallelism results in \textbf{redundant} computation
Parallel processing of **non-overlapping** window data?

Slide-based parallelism limits degree of parallelism.

combine partial results
Idea: Parallelise over data chunks that are **best for hardware**
- Should be independent of **query**!

Task contains one or more **window fragments**
Idea: **Reassemble** partial results to obtain **overall result**

Partial result assembly must also be done in **parallel**
SABER's Hybrid Execution Model

Idea: Permits task execution on all **heterogeneous processors** (eg CPUs, GPUs, ...)
- Scheduler assigns tasks to idle processors on-the-fly

![Diagram of task execution]
SABER Big Data Engine

Dispatching stage
Dispatch fixed-size tasks

Scheduling & execution stage
Dequeue tasks based on HLS

Result stage
Merge & forward partial window results

Java 15K LOC
C & OpenCL 4K LOC

Peter Pietzuch - Imperial College London
What is SABER's Performance?

SABER achieves aggregate performance of CPUs and GPUs
2. Expressing Machine Learning Algorithms

Data filtering → Complex pattern matching → Data queries → Online machine learning, data mining

Increasing complexity

T1(a, b, c)
T2(c, d, e)
T3(g, i, h)
Democratise Big Data Analytics

Growth of Data vs. Growth of Data Analysts

Stored Data accumulating at 28% annual growth rate
Data Analysts in workforce growing at 5.7% growth rate

Data Analyst shortage

美味しい Need to enable more users to do big data analytics…
Programming Models For Big Data Apps?

Declarative query languages (e.g., SQL) challenging for complex algorithms (e.g., iterative machine learning algorithms)
- Typically need to rely on user-defined functions (UDFs)

Distributed dataflow frameworks favour functional programming models
- MapReduce, SQL, PIG, DryadLINQ, Spark, …
- Simplifies consistency and fault tolerance

Domain experts tend to write imperative programs
- Java, Matlab, C++, R, Python, Fortran, …
Abstractions for Machine Learning

Online recommender system

- Recommendations based on past user ratings
- Eg based on collaborative filtering (cf Netflix, Amazon, …)

Executed as dataflow graph
(eg Hadoop, Spark, Flink, …)
Matrix userInput = new Matrix();
Matrix coOcc = new Matrix();

void addRating(int user, int item, int rating) {
    userInput.setElement(user, item, rating);
    updateCoOccurrence(coOcc, userInput);
}

Vector getRec(int user) {
    Vector userRow = userInput.getRow(user);
    Vector userRec = coOcc.multiply(userRow);
    return userRec;
}
// Build the recommendation model using ALS
int rank = 10;
int numIterations = 20;
MatrixFactorizationModel model = ALS.train(JavaRDD.toRDD(ratings), rank, numIterations, 0.01);

// Evaluate the model on rating data
JavaRDD<Tuple2<Object, Object>> userProducts = ratings.map(new Function<Rating, Tuple2<Object, Object>>() {
    public Tuple2<Object, Object> call(Rating r) {
        return new Tuple2<Object, Object>(r.user(), r.product());
    }
});
JavaPairRDD<Tuple2<Integer, Integer>, Double> predictions = JavaPairRDD.fromJavaRDD(model.predict(JavaRDD.toRDD(userProducts)).toJavaRDD().map(new Function<Rating, Tuple2<Tuple2<Integer, Integer>, Double>>() {
    public Tuple2<Tuple2<Integer, Integer>, Double> call(Rating r){
        return new Tuple2<Tuple2<Integer, Integer>, Double>(new Tuple2<Integer, Integer>(r.user(), r.product()), r.rating());
    }
}));
JavaRDD<Tuple2<Double, Double>> ratesAndPreds = JavaPairRDD.fromJavaRDD(ratings.map(new Function<Rating, Tuple2<Tuple2<Integer, Integer>, Double>>() {
    public Tuple2<Tuple2<Integer, Integer>, Double> call(Rating r){
        return new Tuple2<Tuple2<Integer, Integer>, Double>(new Tuple2<Integer, Integer>(r.user(), r.product()), r.rating());
    }
})).join(predictions).values();
Collaborative Filtering in Spark (Scala)

- Build the recommendation model using ALS
  ```scala
  val rank = 10
  val numIterations = 20
  val model = ALS.train(ratings, rank, numIterations, 0.01)
  ```

- Evaluate the model on rating data
  ```scala
  val usersProducts = ratings.map {
    case Rating(user, product, rate) => (user, product)
  }
  val predictions =
    model.predict(usersProducts).map {
      case Rating(user, product, rate) => ((user, product), rate)
    }
  val ratesAndPreds = ratings.map {
    case Rating(user, product, rate) => ((user, product), rate)
  }.join(predictions)
  ```

- All data is immutable, no fine-grained model updates
State elements (SEs) are mutable in-memory data structures
- Tasks have local access to SEs
- SEs can be shared between tasks
Stateful Dataflow Graphs (SDG)

Annotated Java program
(@Partitioned, @Partial, @Global, …)

Program.java

Static program analysis

Data-parallel Stateful Dataflow Graph (SDG)

SEEP distributed dataflow framework

Translation & checkpoint-based fault tolerance

Cluster
Conclusions

Big Data Systems have enabled the Big Data revolution
- Driven by advances in data centre technology and parallel hardware

1. Performance means exploiting new parallel hardware (GPUs)
- Must decouple query semantics from hardware parameters

👉 SABER engine: Hybrid CPU/GPU execution model

2. Usability means providing right programming abstractions
- Imperative programming models still going strong

👉 SDG: Stateful Big Data processing for machine learning
Acknowledgements: LSDS Group

We're recruiting: PhDs, Post-Docs

Thank you! Any Questions?

Peter Pietzuch
<prp@doc.ic.ac.uk>
http://lsds.doc.ic.ac.uk