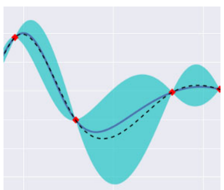
Large-scale Data Processing and Optimisation

Overview



Eiko Yoneki

University of Cambridge Computer Laboratory

Massive Data: Scale-Up vs Scale-Out

- Popular solution for massive data processing
  → scale and build distribution, combine theoretically unlimited number of machines in single distributed storage
  → Parallelisable data distribution and processing is key
- Scale-up: add resources to single node (many cores) in system (e.g. HPC)
- Scale-out: add more nodes to system (e.g. Amazon EC2)

2

Technologies supporting Cluster Computing

- Distributed infrastructure
  - Cloud (e.g. Infrastructure as a service, Amazon EC2, GCP, Azure)
- Storage
  - Distributed storage (e.g. Amazon S3, Hadoop Distributed File System (HDFS), Google File System (GFS))
- Data model/indexing
  - High-performance schema-free database (e.g. NoSQL DB - Redis, BigTable, Hbase, Neo4J)
- Programming model
  - Distributed processing (e.g. MapReduce)

3

Data Processing Stack

Data Processing Layer

Streaming Processing
Storm, SEEP, Naiad, Spark Streaming, Flink, Milwheel, Google Dataflow...

Query Language
Pig, Hive, SparkSQL, DryadLINQ...

Machine Learning
Rllib, Caffe, Keras, Torch, MLlib...

Execution Engine
MapReduce, Spark, Tensorflow, Ray, Flumejava...

Graph Processing
Pregel, Giraph, GraphLab, PowerGraph, (Dato), GraphX, X-Stream...

Storage Layer

Distributed File Systems
GFS, HDFS, Amazon S3, Flat FS..

Operational Store/NoSQL DB
Big Table, Hbase, Dynamo, Cassandra, Redis, Mongo, Spanner...

Logging System/Distributed Messaging Systems
Kafka, Flume...

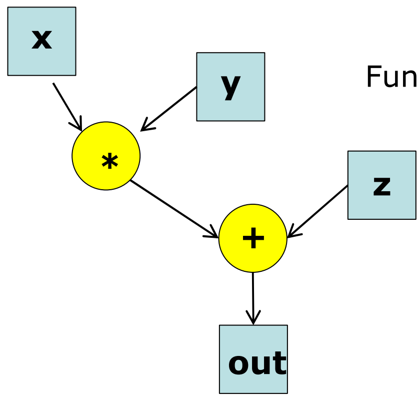
Resource Management Layer

Resource Management Tools
Mesos, YARN, Borg, Kubernetes, EC2, OpenStack...

4

Data Flow Programming

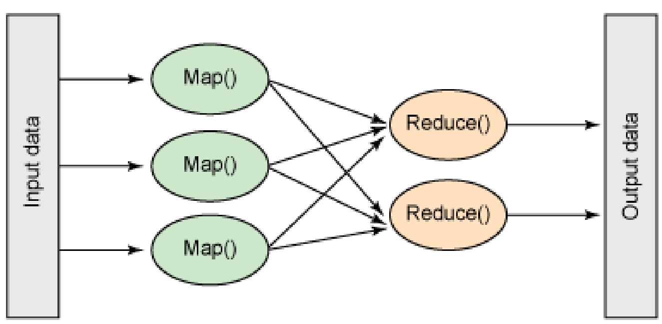
- Non-standard programming models
- Powerful abstraction: mapping computation into dataflow graphs



Function f(x, y, z) = x* y + z

MapReduce Programming

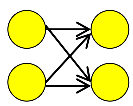
- Target problem needs to be parallelisable
- Split into a set of smaller code (map)
- Next small piece of code executed in parallel
- Results from map operation get synthesised into a result of original problem (reduce)



Data Flow Programming Examples

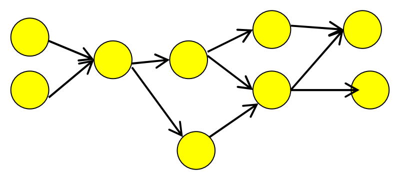
- Data (flow) parallel programming
  - e.g. MapReduce, Dryad/LINQ, NAIAD, Spark, Tensorflow…
  - e.g. ML compiler: computation graphs

MapReduce:
Hadoop

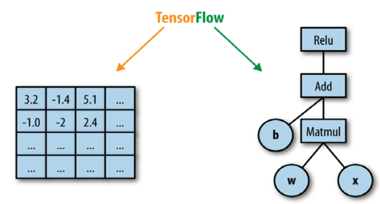


Two-Stage fixed dataflow

DAG (Directed Acyclic Graph)
based: Dryad/Spark…



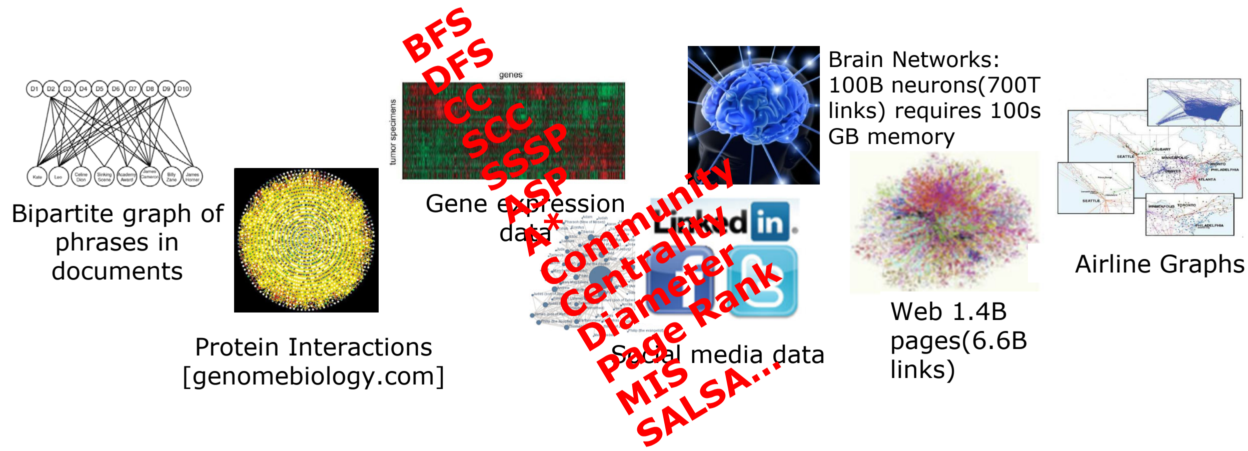
More flexible dataflow model



Data Flow Programming's Usefulness for ML

- Data flow programming holds the key to natural, modular, streamlined ML specification
- Same language for layer, network, and integration specification
  - For complex NN classes, RL with complex reward, gated networks, attention/transformers, mix of NN and data processing
- Same language for ML algorithmics and compiler design
- Well-established analysis and compilation techniques integrated within existing ML compilers (e.g. MLIR)
  - MLIR (Multi-Level Intermediate Representation) is intended to be a hybrid IR which can support multiple different requirements in a unified infrastructure (https://mlir.llvm.org/)

Emerging Massive-Scale Graph Data



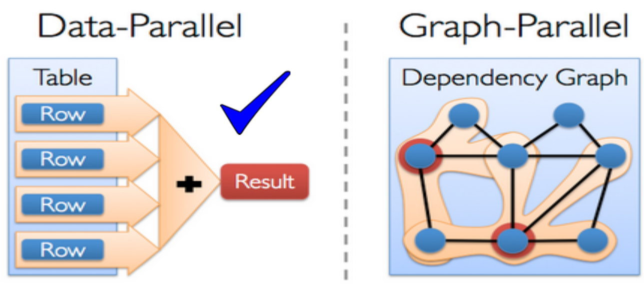
Graph Computation Challenges

1. Graph algorithms (BFS, Shortest path)
2. Query on connectivity (Triangle, Pattern)
3. Structure (Community, Centrality)
4. ML & Optimisation (Regression, SGD)

- Data driven computation: dictated by graph's structure and parallelism based on partitioning is difficult
- Poor locality: graph can represent relationships between irregular entries and access patterns tend to have little locality
- High data access to computation ratio: graph algorithms are often based on exploring graph structure leading to a large access rate to computation ratio

Data-Parallel vs. Graph-Parallel

- *Data-Parallel* for all? *Graph-Parallel* is hard!
  - Data-Parallel (sort/search - randomly split data to feed MapReduce)
  - Not every graph algorithm is parallelisable (interdependent computation)
  - Not much data access locality
  - High data access to computation ratio



Graph-Parallel

- Graph-Parallel (Graph Specific Data Parallel)
  - Vertex-based iterative computation model
  - Use of iterative Bulk Synchronous Parallel Model
    → Pregel (Google), Giraph (Apache), Graphlab, GraphChi (CMU - Dato)
  - Optimisation over data parallel
    → GraphX/Spark (U.C. Berkeley)
  - Data-flow programming – more general framework
    → NAIAD (MSR), TensorFlow…

Bulk synchronous parallel: Example

- Finding the largest value in a connected graph

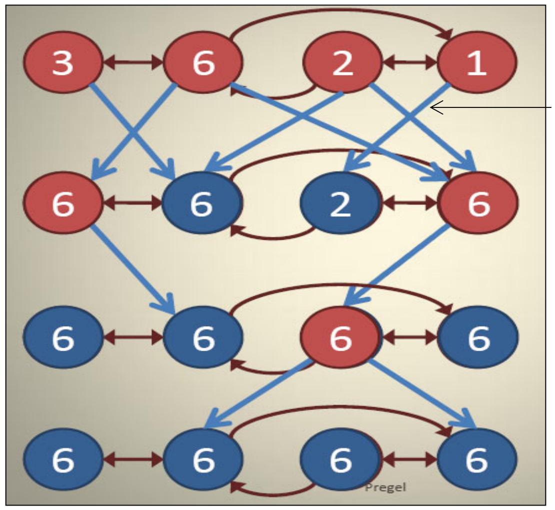
Local Computation

Communication

Local Computation

Communication

...

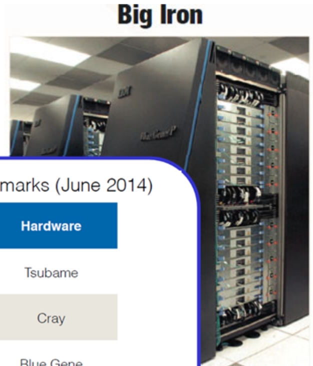


Message

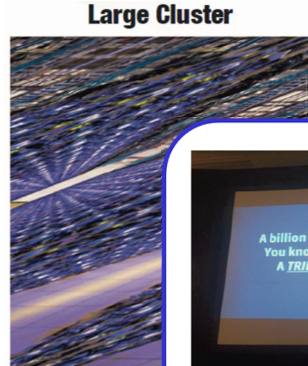
Are Large Clusters and Many cores Efficient?

- Brute force approach really efficiently works?
  - Increase of number of cores (including use of GPU)
  - Increase of nodes in clusters

Big Iron

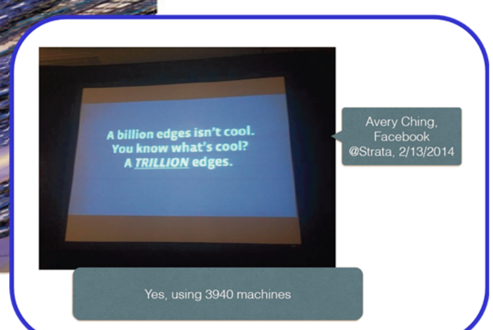


Large Cluster

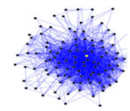


HPC/Graph500 benchmarks (June 2014)

Graph Edges	Hardware
1 trillion	Tsubame
1 trillion	Cray
1 trillion	Blue Gene
1 trillion	NEC



Do we really need large clusters?



- Laptops are sufficient?

Twenty pagerank iterations

System	cores	twitter_rv	uk_2007_05
Spark	128	857s	1759s
Giraph	128	596s	1235s
GraphLab	128	249s	833s
GraphX	128	419s	462s
Single thread	1	300s	651s

Label propagation to fixed-point (graph connectivity)

System	cores	twitter_rv	uk_2007_05
Spark	128	1784s	8000s+
Giraph	128	200s	8000s+
GraphLab	128	242s	714s
GraphX	128	251s	800s
Single thread	1	153s	417s

Fixed-point iteration: All vertices active in each iteration (50% computation, 50% communication)

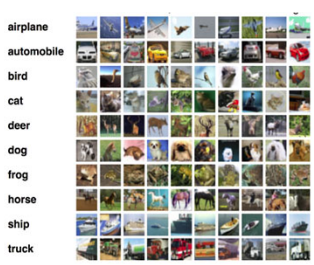
Traversal: Search proceeds in a frontier (90% computation, 10% communication)

from Frank McSherry HotOS 2015

Data Processing Paradigm Change

- Emergence of modern Neural Networks Applications
- Practicalities of training Neural Networks
- Leveraging heterogeneous hardware
- Traditional dataflow programming does not deal with mathematical objects (no deep learning back then), now control flow requires to be numerically differentiable (i.e. TensorFlow)

Image Classification



Reinforcement Learning

Image courtesy: The Guardian

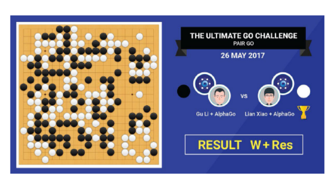


Image courtesy: Twitter - Deep Mind AI

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Challenging: Computer Systems Optimisation

- How do we improve performance:
  - Manual tuning
  - Auto-tuning
- What is performance? – objective function of optimisation
  - Resource usage (e.g. time, power)
  - Computational properties (e.g. accuracy, fairness, latency)
  - Large number of parameters
  - Evaluation is slow and expensive
- What is Optimisation Model?
  - Short-term dynamic control (e.g. stream processing: distinct workload or dynamic workload)
  - Combinatorial optimisation (e.g. indexing DB, device assignment)

Many systems problems are combinatorial in nature

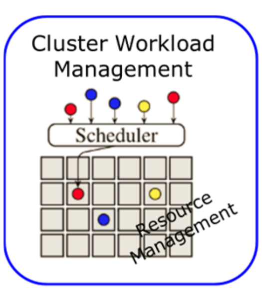
17

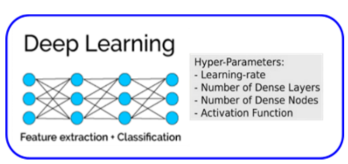
UNIVERSITY OF CAMBRIDGE
Computer Laboratory

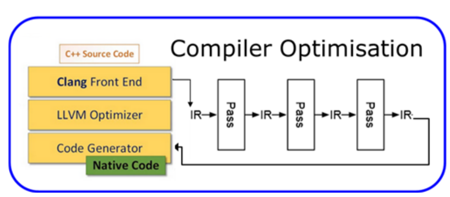
Tuning Computer Systems is Complex

- Complex and High Dimension Parameter Space
- Expensive Objective Functions
- Hand-crafted solutions impractical, often left static or configured through extensive offline analysis
- Not well-tuned system's performance does not scale

→ Necessity of scalable Auto-tuning







18

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Machine Learning and Optimisation

- Function Optimisation
  - Find the set of inputs to a target objective function that result in the minimum or maximum of the function
- Function Approximation:
  - Generalise from specific examples to a reusable mapping function for making predictions on new examples
  - ML can be described as function approximation as approximating the unknown underlying function that maps examples of inputs to outputs in order to make predictions on new data
  - Function approximation often uses function optimisation
- At the core of many ML algorithms is an optimisation algorithm!

19

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Optimisation: Iterative Operation

- Common to use an iterative global search algorithm for optimisation problem
- e.g. Bayesian optimisation algorithm that is capable of simultaneously approximating the target function that is being optimised while optimising it.
- Automated machine learning (AutoML) algorithms being used to choose an algorithm, an algorithm and hyperparameters, or data preparation, algorithm and hyperparameters, with very little user intervention

20

UNIVERSITY OF CAMBRIDGE Computer Laboratory

Massive Parameter Space in Large-scale Systems

Parameter	Granularity
Number of cluster nodes	Coarse
ML Hyperparameters	↕
Compiler Flags	Fine

Optimise parameters θ?

Minimise some cost function f(θ)

...where cost is runtime, memory, I/O, or combination of all...

- Many dimensions
- Expensive objective function
- Hand-crafted solutions impractical (e.g. extensive offline analysis)

➡ Blackbox Optimisation

UNIVERSITY OF CAMBRIDGE Computer Laboratory

Search Parameter Space

Random search: No risk of 'getting stuck' potentially many samples required

PetaBricks

Evolution strategies: Evaluate permutations against fitness function

Hill Climbing OpenTuner

SPEARMINT

Bayes Opt: Sample efficient, requires continuous function, some configuration

Random Search	Genetic algorithm / Simulated annealing	Bayesian Optimisation
No overhead	Slight overhead	High overhead
High #evaluation	Medium-high #evaluation	Low #evaluation

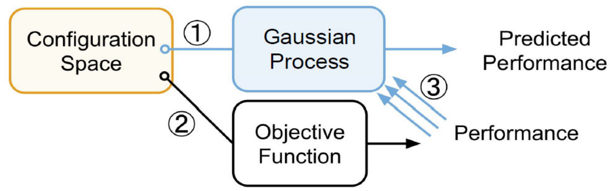
22

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Bayesian Optimisation

- Iteratively builds probabilistic model of objective function
- Typically Gaussian process as probabilistic model
- Data efficient: converges quickly

Input: Objective function $f()$
Input: Surrogate function initial distribution G
Input: Acquisition function $a()$
for $i = 1, 2, \ldots$ **do**
  Sample point: $\boldsymbol{x}_t \leftarrow \arg\max_{\boldsymbol{x}} a(G, \boldsymbol{x})$
  Evaluate new point: $y_t \leftarrow f(\boldsymbol{x}_t)$
  Update surrogate distribution: $G \leftarrow G \mid (\boldsymbol{x}_t, y_t)$
end for



Pros:
- ✓ Data efficient: converges in few iterations
- ✓ Able to deal with noisy observations

Cons:
- ✗ In many dimensions, model does not converge to the objective function

① Find promising point (high performance value in the model)
② Evaluate the objective function at that point
③ Update the model to reflect this new measurement

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Surrogate Model in Bayesian Optimisation

Table 2.1: Comparison of surrogate models for BO

Model	Advantages	Disadvantages
Parametric models	• Quickly fit long-distance trends	• Require known structure of f
Gaussian processes	• Expressive • Flexible	• Fitting is $O(n^3)$ in train-data size • Continuous, non-hierarchical configuration space only
Tree-Parzen estimators	• Fitting is $O(n)$ in train-data size • Categorical and hierarchical configuration space supported	• Less sample efficient than GP
Random forests	• Computationally very cheap • Categorical and hierarchical configuration space supported	• Inaccurately extrapolates uncertainty

- Structural information (e.g. DAG model) improves Optimisation.

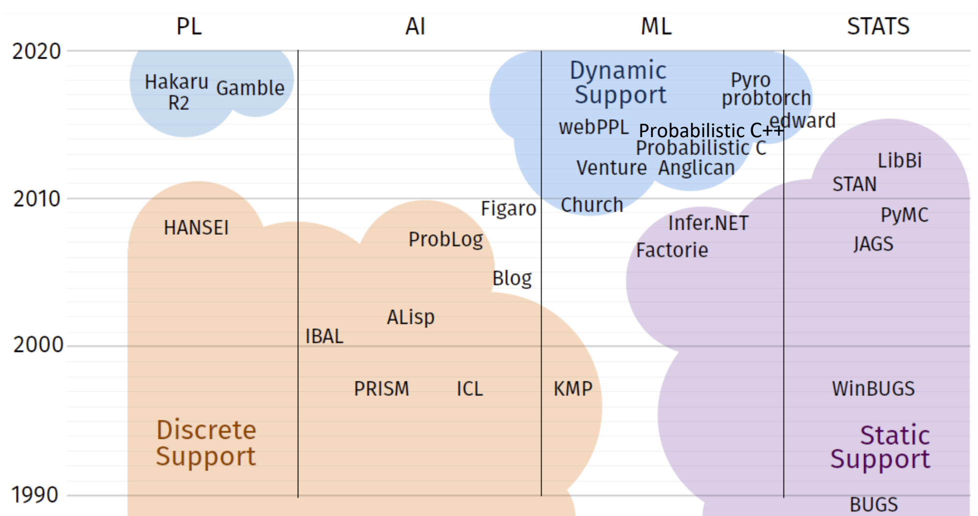
Probabilistic Model

- Probabilistic models incorporate random variables and probability distributions into the model
  - Deterministic model gives a single possible outcome
  - Probabilistic model gives a probability distribution
- Used for various probabilistic logic inference (e.g. MCMC-based inference, Bayesian inference…)

Tutorial: Session 5 – Guest Lecture by Hong Ge

25

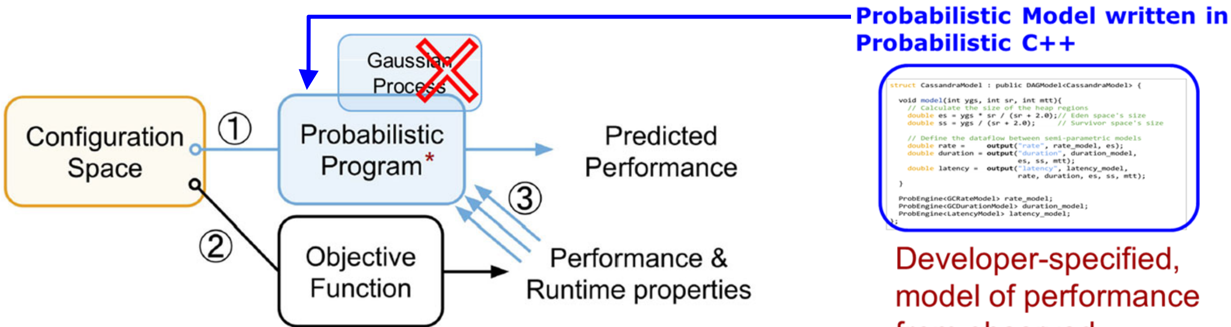
Probabilistic Programming



B. Paige

26

Structured Bayesian Optimisation (SBO)



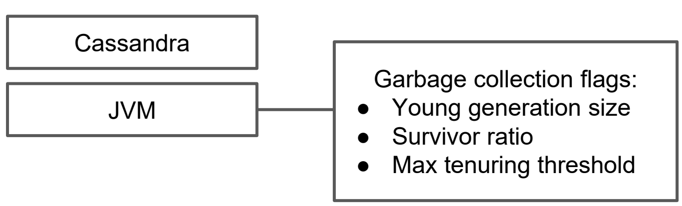
Developer-specified, model of performance from observed performance + arbitrary runtime characteristics

✓ Better convergence
✓ Use all measurements

BOAT: a framework to build **B**esp**O**ke **A**uto-**T**uners

Example: JVM Garbage Collection

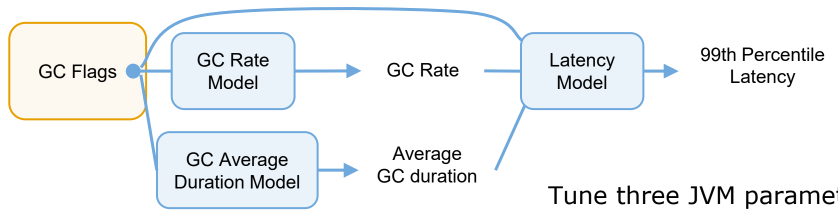
- Cassandra's garbage collection



- Minimise 99th percentile latency of Cassandra

Performance Improvement from Structure

User-given probabilistic model structured in semi-parametric model using Directed Acyclic Graph

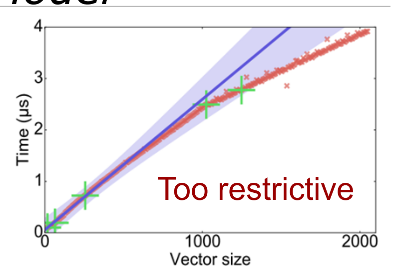


Tune three JVM parameters of database (Cassandra) to minimise latency

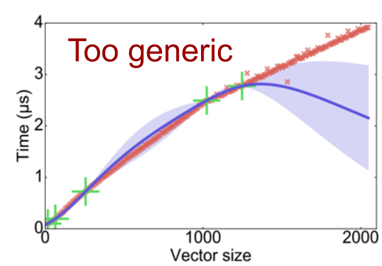
29

Semi-parametric Model

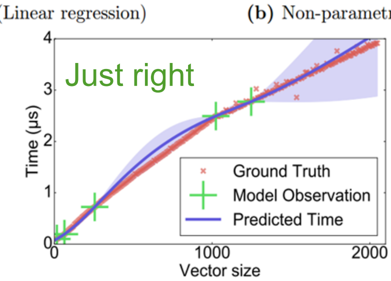
- Easy to use and well suited to SBO
  - Understand general trend of Objective function
  - High precision in region of optimum for finding highest performance



(a) Parametric (Linear regression)



(b) Non-parametric (Gaussian process)



(c) Semi-parametric (Combination)

30

DAG model in BOAT

```
struct CassandraModel : public DAGModel<CassandraModel> {

  void model(int ygs, int sr, int mtt){
    // Calculate the size of the heap regions
    double es = ygs * sr / (sr + 2.0);// Eden space's size
    double ss = ygs / (sr + 2.0);     // Survivor space's size

    // Define the dataflow between semi-parametric models
    double rate =     output("rate", rate_model, es);
    double duration = output("duration", duration_model,
                             es, ss, mtt);
    double latency =  output("latency", latency_model,
                             rate, duration, es, ss, mtt);
  }

  ProbEngine<GCRateModel> rate_model;
  ProbEngine<GCDurationModel> duration_model;
  ProbEngine<LatencyModel> latency_model;
};
```

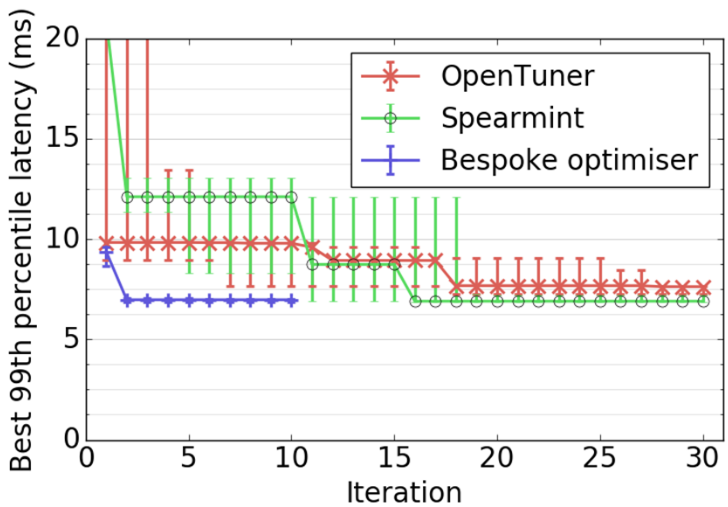
31

GC Rate Semi-parametric model

```
struct GCRateModel : public SemiParametricModel<GCRateModel> {

  GCRateModel() {
    allocated_mbs_per_sec =
     std::uniform_real_distribution<>(0.0, 5000.0)(generator);
    // set the GP parameters here
  }

  double parametric(double eden_size) const {
    // Model the rate as inversely proportional to Eden's size
    return allocated_mbs_per_sec / eden_size;
  }

  double allocated_mbs_per_sec;
};
```

32

Evaluation: Garbage collection

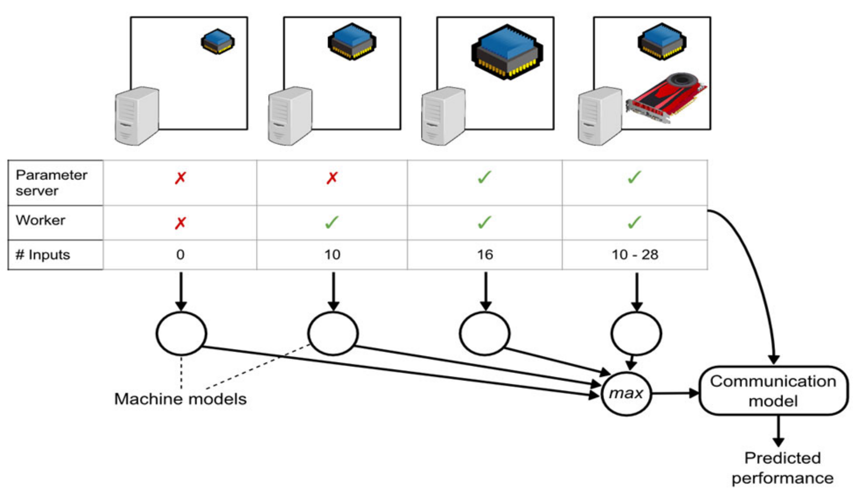


33

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

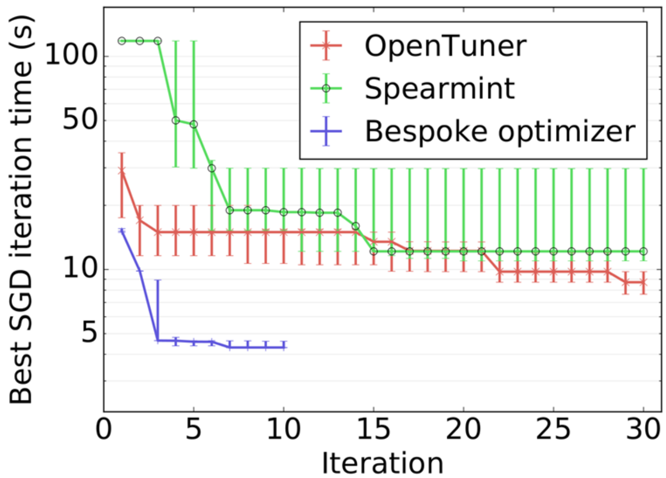
Distributed Scheduling of Neural Networks (SGD)

- Tune scheduling over 10 machines, setting ~30 parameters (e.g. ~10^{53} possible valid configurations)



34

Evaluation: Neural network scheduling



Default configuration: 9.82s
OpenTuner: 8.71s
BOAT: **4.31s**

Existing systems don't converge!

35

Further Bayesian Optimisation…

- BO overview/Tutorial
  - https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2024_2025/aid/BO_overview_Archambeau.pdf
  - https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2024_2025/aid/BO_overview_adams.pdf
  - https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2024_2025/aid/BO_overview_gonzalez.pdf
- Papers
  - Review paper by Shahriari, et al. (2016): Taking the Human Out of the Loop: A Review of Bayesian Optimization. Proceedings of the IEEE 104(1):148-175, 2016.
  - Slides by Ryan Adams (2014): A Tutorial on Bayesian Optimization for Machine Learning. CIFAR NCAP Summer School.
  - Slides by Peter Frazier (2010): Tutorial: Bayesian Methods for Global and Simulation Optimization. INFORMS Annual Meeting.

36

Auto-Tuning

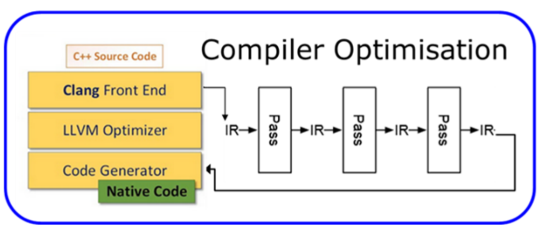
- Manual Tuning
  - User to learn expert knowledge and not transferable
  - e.g. Ottertune (manually selects limited number of parameters then use BO)
- Automated Tuning
  - Divide-and-diverge sampling to explore the configuration space
  - Use of Gaussian processes, but it struggles to make accurate performance predictions because of high dimensionality

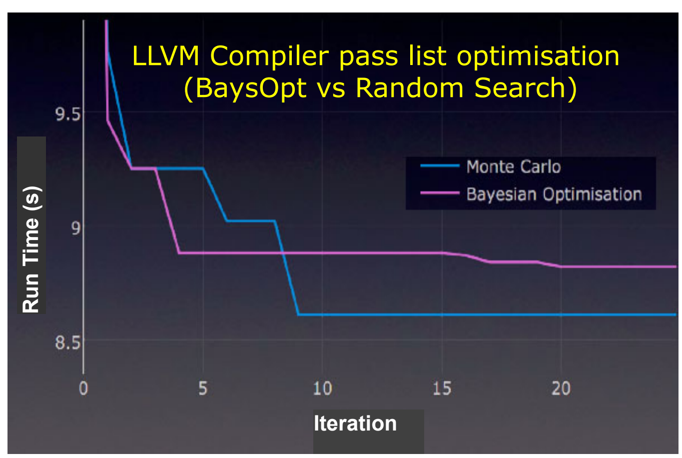
→Generic Auto-Tuning with DAG models

- Use of DAG models for surrogate model, which mitigates the curse of dimensionality while also retaining all configurable variables
- Exploit data analysis to identify parameter dependencies
- Automatic building of DAG models: use of Bayesian Networks

37

Bayesian Optimisation not for Combinatorial Model

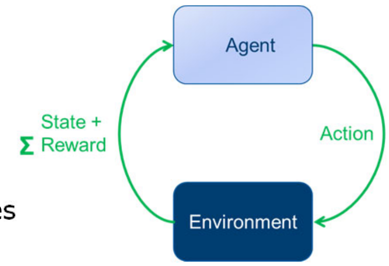




38

Reinforcement Learning in Computer Systems

- Agent interacts with Dynamic environment
- Goal: Maximise expectations over rewards in agent's lifetime
- Notion of Planning/Control, not single static configuration



What makes RL different from other ML paradigms?

- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential)
- Agent's actions affect the subsequent data it receives

Practical Consideration:

- Action spaces do not scale
- Exploration in production system not a good idea
- Simulations can oversimplify problem (Expensive to build)
- Online steps take too long

39

Reinforcement Learning for Optimisation

Many problems in systems are sequential decision making and/or combinatorial problems

- Compiler Optimisation
- Chip placement
- Datacentre resource allocation
- Network congestion control with multiple connections
- Wide range of signals to make decisions (e.g., VM allocation)
- Database: Query optimiser, Dynamic indexing…

40

A brief history of Deep Reinforcement Learning Tools

Gen (2014-16): Loose research scripts (e.g. DQN), high expertise required, only specific simulators

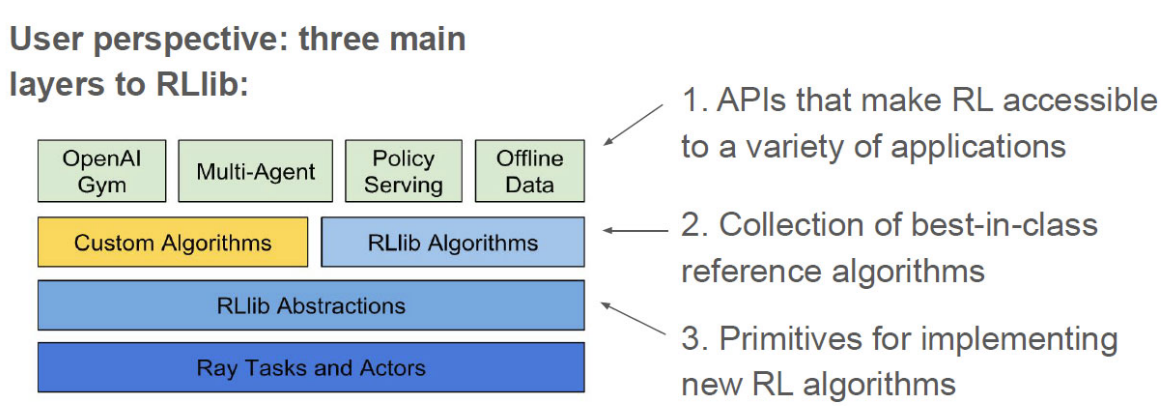
Gen (2016-17): OpenAI gym gives unified task interface, reference implementations

- Good results on some environments (e.g. game), difficult to retool to new domains and execution modes
- Abstractions/Libraries: not fully reusable, customised towards game simulators
- High implementation risk: lack of systematic testing, performance strongly impacted by noisy heuristics

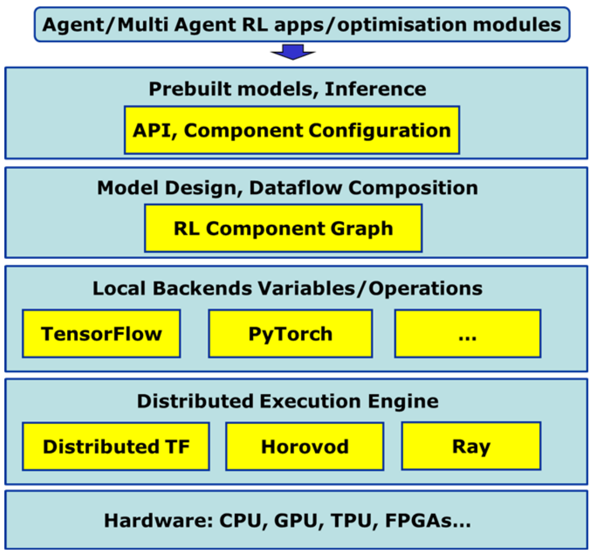
Gen (2017-): Generic declarative APIs, distributed abstractions (Ray Rllib, RLGraph), some standard *flavours* emerge

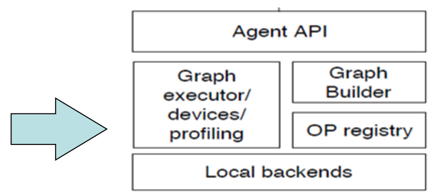
Still Problems... Tightly coupled execution/logic, testing, reuse...

RLlib (UC Berkeley) Architecture



RLgraph: Modular Dataflow Composition





- ... is a programming model to design and execute RL algorithms across execution paradigms
- ... generates incrementally testable, transparently configurable code through a staged build process

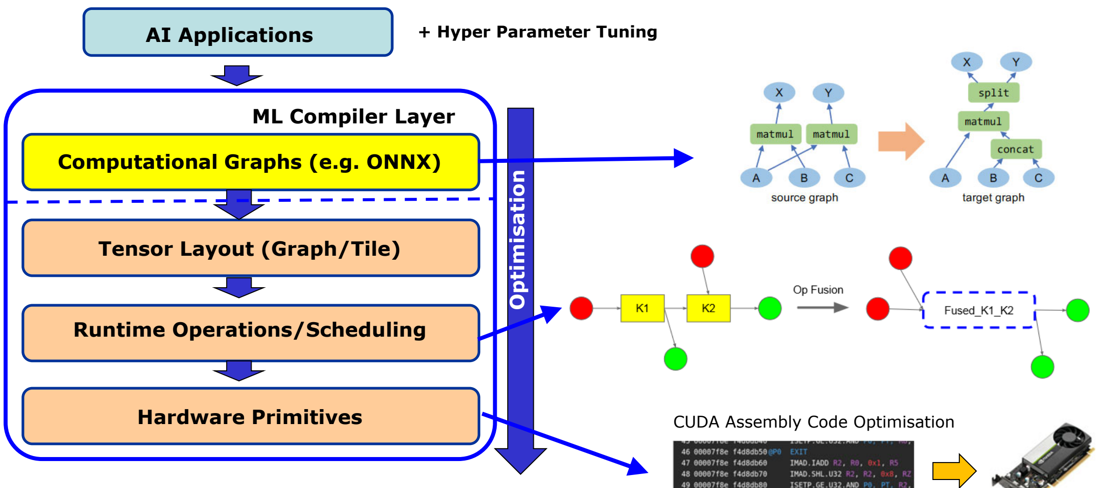
43

ML Compiler Challenges

- Fast tensor operation capable compilers: Complex optimisation plays a crucial role for LLM processing
  - Use of ML to optimise ML Compiler
  - Superoptimisation: Deal with massive parameter space
    → use of Equality Saturation (ES), MCTS
    → use of reinforcement Learning
  - Many compiler optimisations are phase ordering problem
- Challenge to bridge the gap between ML models and HW
- Existing ML Compiler
  - Apache TVM
  - NVIDIA TensorRT - CuDNN
  - ONNX runtime
  - LLVM
  - Google MLIR
  - TensorFlow XLA
  - Meta Glow
  - PyTorch nvFuser
  - INTEL pLAIDml
  - Open VINO

9

ML Compiler Optimisation



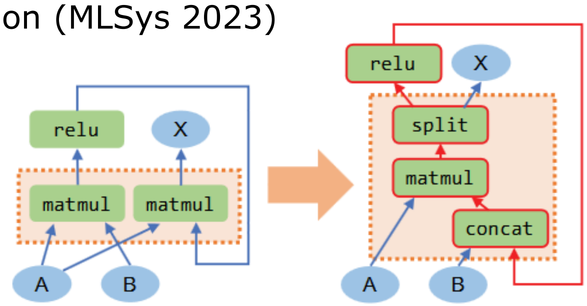
TensorFlow, MXNet, and PyTorch

See ML Compiler Tutorial: https://mlc.ai/summer22/schedule
Also Survey: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9222299

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Optimising DNN Computation with Graph Substitutions

- TASO (SOSP, 2019): Performance improvement by transformation of computation graphs
- PET (OSDI, 2021): Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections
- Equality Saturation for Tensor Graph Superoptimization (MLSys 2021)
- X-RLflow: Graph Reinforcement Learning for Neural Network Subgraph Transformation (MLSys 2023)



(a) before substitution (b) after substitution

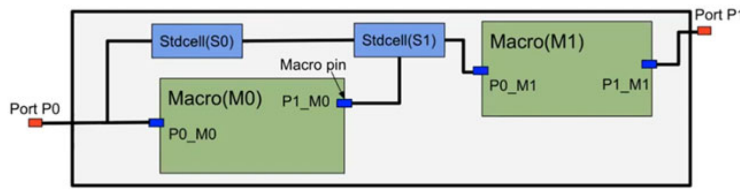
UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Chip Placement with Reinforcement Learning

- A. Mirhoseini and A. Goldie: A graph placement methodology for fast chip design, Nature, 2021.

Partitioning-Based Methods (e.g. MinCut)	Stochastic/Hill-Climbing Methods (e.g. Simulated Annealing)
Analytic Solvers (e.g. RePlAce)	Learning-Based Methods

- A form of graph resource optimization
- Place the chip components to minimize the latency of computation, power consumption, chip area and cost, while adhering to constraints, such as congestion, cell utilization, heat profile, etc.



47

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Summary: Massive Data Processing and Optimisation

→Dataflow is key element used in optimisation

→Parameter space is complex, large and dynamic/combinatorial

- Systems are nonlinear and difficult to model manually → Exploit ML
- Reinforcement Learning to optimise dynamic combinatorial problem
- Key concept behind is Dataflow (~=Computational Graph) structural transformation/Decomposition

→Exploit structural information for model decomposition to accelerate optimisation process and/or transform the structure

→Bayesian Optimisation and Reinforcement Learning are key

48

Course Schedule

https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2023_2024

Session 1: Introduction

Session 2: Data Flow Programming: Map/Reduce to TensorFlow to ML

Session 3: Large-scale Graph Data Processing

Session 4: Hands-on Tutorial: Distributed Data Flow Programing

Session 5: Probabilistic Programming + BO : Guest lecture (Hong Ge)

Session 7: Optimisation of Computer Systems (HW design, DB…)

Session 6: Optimisation in ML Compiler (Superoptimisation…)

Session 8: Project Study Presentation