To Petascale and Beyond: Apache Flink in the Clouds

Greg Hogan
greg@apache.org

My Project Background

- Apache Flink Committer since 2016
- Progressing from Hadoop MapReduce

  enticed by tuples, fluent API, and scalability

  enduring for community and growth
- Commits for performance and scalability
- Contributed Gelly graph generators and native algorithms

Agenda

- [x] Graphs with Gelly
- [ ] Native algorithms
- [ ] Configuring at scale
- [ ] Sort benchmark
- [ ] Profiling and benchmarking
- [ ] In development

Why batch? Why Gelly?

- Batch

  strong similarities between batch and streaming

  benchmarking is in essence batch

- Gelly

  diverse, scalable algorithms

  that stress distributed architectures

What is Gelly?

Graph API:

Graph, Vertex, Edge, … BipartiteGraph, BipartiteEdge (FLINK-2254)

```
public class Graph<K, VV, EV> {
  private final ExecutionEnvironment context;
  private final DataSet<Vertex<K, VV>> vertices;
  private final DataSet<Edge<K, EV>> edges;


public class Vertex<K, V> extends Tuple2<K, V> {


public class Edge<K, V> extends Tuple3<K, K, V>{
```

from*, getTriplets, groupReduceOn*, joinWith*, map*, reduceOn*

What is Gelly?

Iterative graph algorithm models:

vertex-centric (pregel)

```
public abstract void compute(Vertex<K, VV> vertex, MessageIterator<Message> messages);
```

scatter-gather (spargel)

```
public abstract void sendMessages(Vertex<K, VV> vertex) throws Exception;
public abstract void updateVertex(Vertex<K, VV> vertex, MessageIterator<Message> inMessages);
```

gather-sum-apply

```
public abstract M gather(Neighbor<VV, EV> neighbor);
public abstract M sum(M arg0, M arg1);
public abstract void apply(M newValue, VV currentValue);
```

What is Gelly?

- Generate scale-free graphs independent of parallelism

  Complete, Cycle, Grid, Hypercube, Path, **RMat**, Star

- Composable building blocks for directed and undirected graphs

  degree annotate, simplify, transform, …

- Algorithms (DataSet) and analytics (accumulators)

- Checksum validation

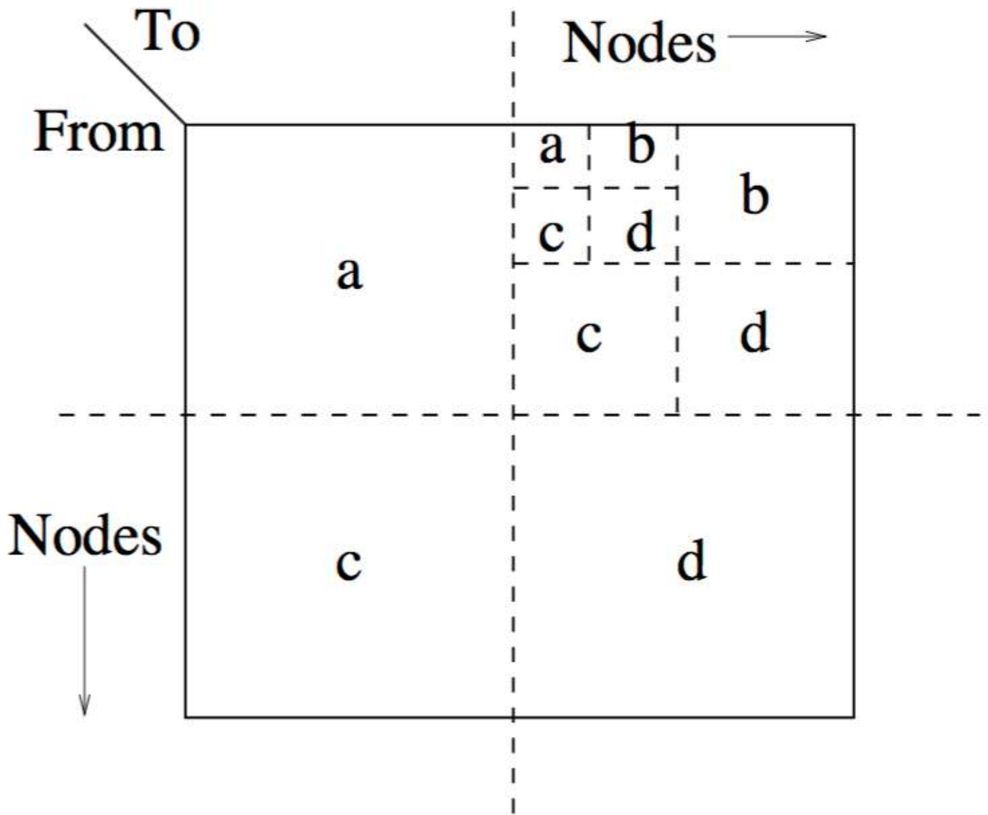
R-MAT: Recursive Matrix

- CMU: Chakrabarti, Zhan, Faloutsos

  power-law degree, “community” structure, small diameter

  parsimony, realism, generation speed

- vertex count: 2^{scale}
- edge count : $2^{scale} * edge\ factor$
- bit probabilities: a + b + c + d = 1
- random noise, clip-and-flip, bipartite, …

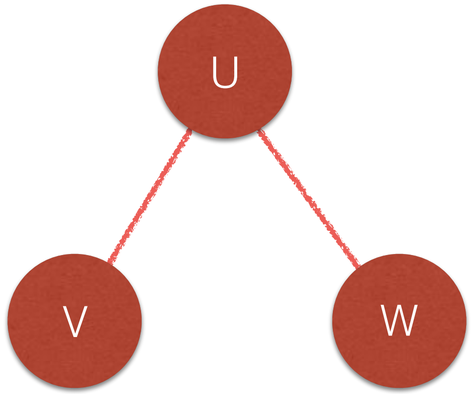


Glossary

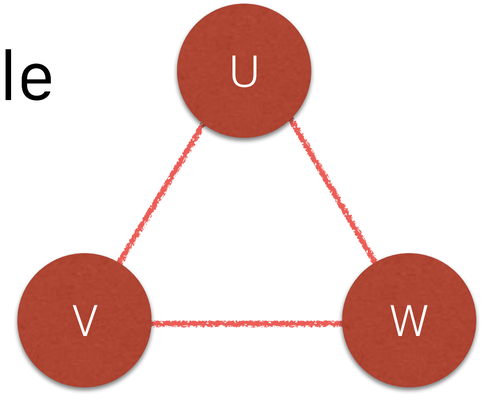
Vertex

Edge

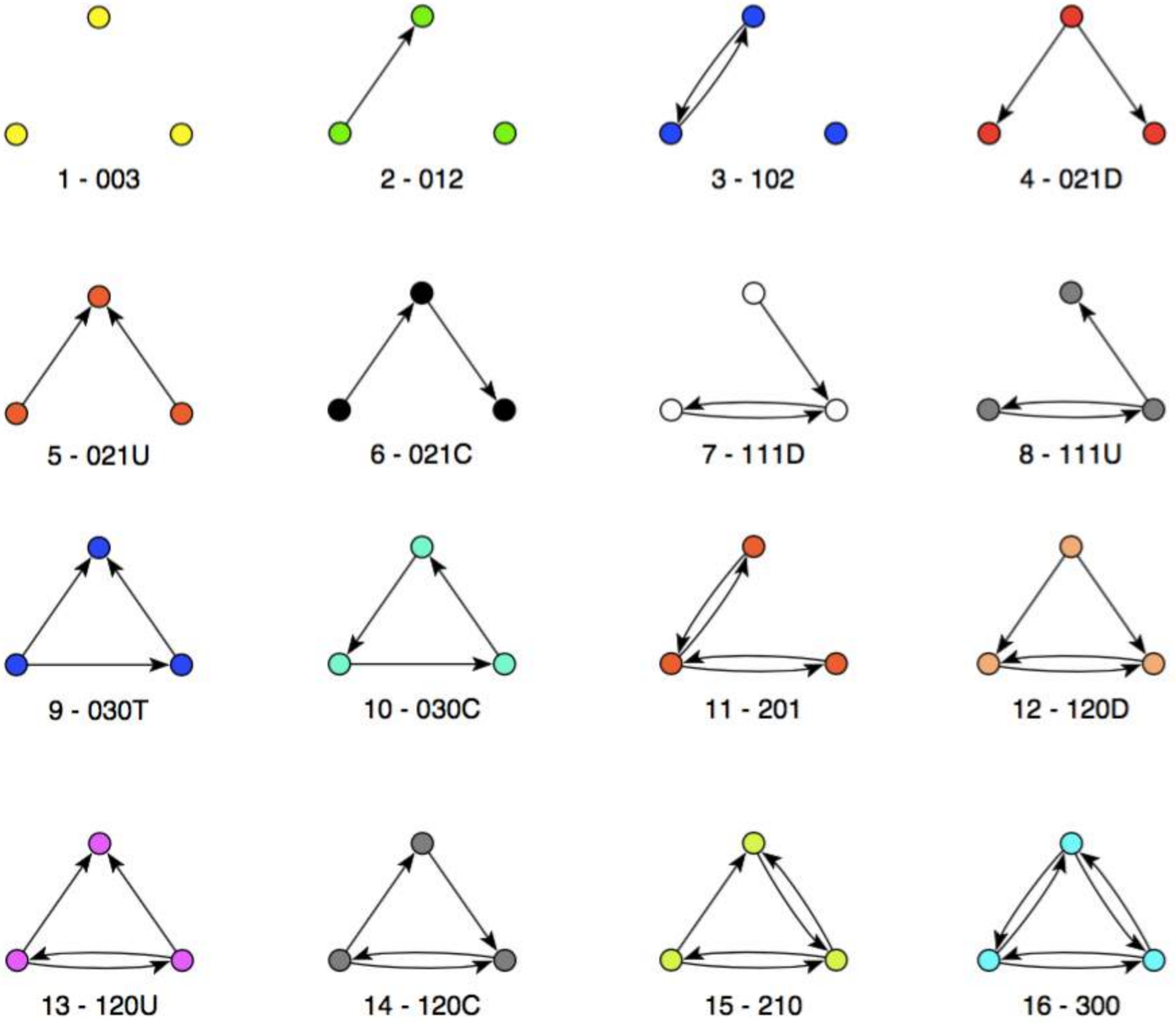
Open Triplet



Closed Triplet / Triangle



Triads


1 - 003
2 - 012
3 - 102
4 - 021D
5 - 021U
6 - 021C
7 - 111D
8 - 111U
9 - 030T
10 - 030C
11 - 201
12 - 120D
13 - 120U
14 - 120C
15 - 210
16 - 300

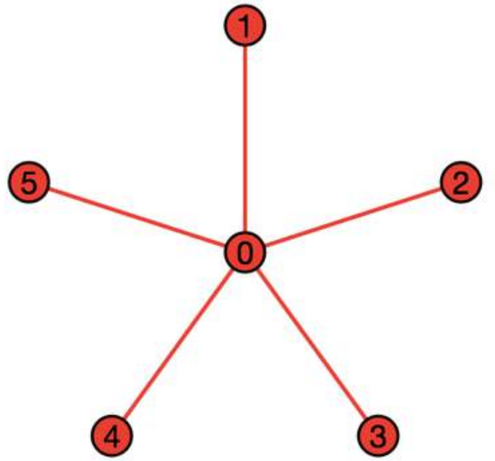
Graph500 Metrics

Scale	Vertices	Edges	Max Degree	△ Triplets	Triplets
16	46,826	955,507	9,608	33,319,057	619,718,882
20	646,203	16,084,983	64,779	1,225,406,293	35,034,305,809
24	8,869,117	263,432,942	406,382	41,744,066,881	1,782,644,764,135
28	121,233,960	4,259,425,690	2,457,580	1,357,806,114,141	84,905,324,934,756
32	1,652,616,155	68,460,935,436	14,455,055	43,031,369,386,717	3,870,783,341,130,190

Agenda

- [x] Graphs with Gelly
- [x] Native algorithms
- [ ] Configuring at scale
- [ ] Sort benchmark
- [ ] Profiling and benchmarking
- [ ] In development

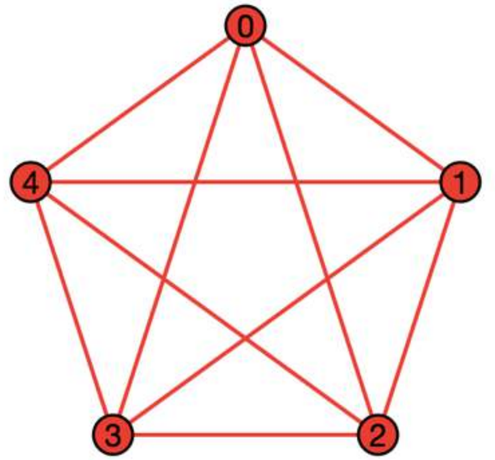
Clustering Coefficient



- Measures connectedness of a vertex’s neighborhood
- Ranges from 0.0 to 1.0:

  0.0: no edges connecting neighbors (star graph or tree)

  1.0: each pair of neighbors is connected (clique)



- Each connection between neighbors forms a triangle

Clustering Coefficient

- Local Clustering Coefficient is fraction of edges between neighbors

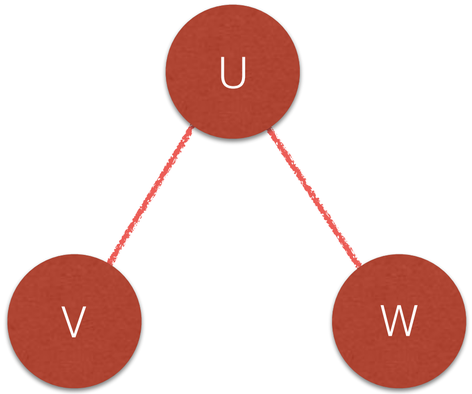
$$LCC_i = \frac{|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{d_i(d_i - 1)}$$

- Global Clustering Coefficient is fraction of connected triplets

$$GCC = \frac{3 \times \text{number of triangles}}{\text{number of connected triplets of vertices}}$$

Enumerating Triangles

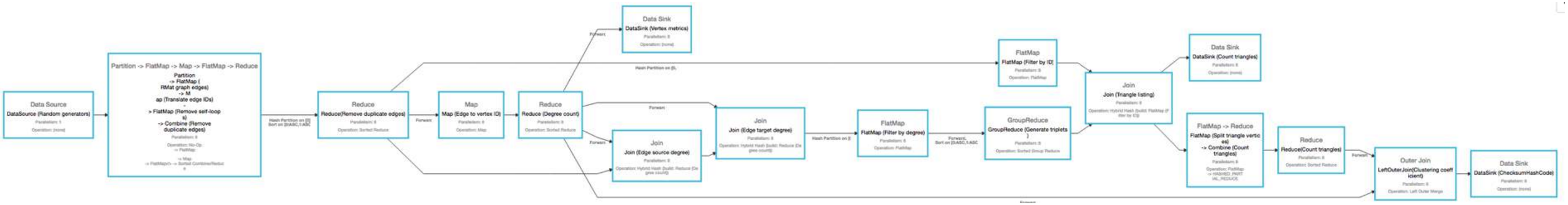
- Thomas Schank: Algorithmic Aspects of Triangle-Based Network Analysis
- Join all edges with all (open) triplets



- Naive algorithm discovers each triangle at all three vertices;

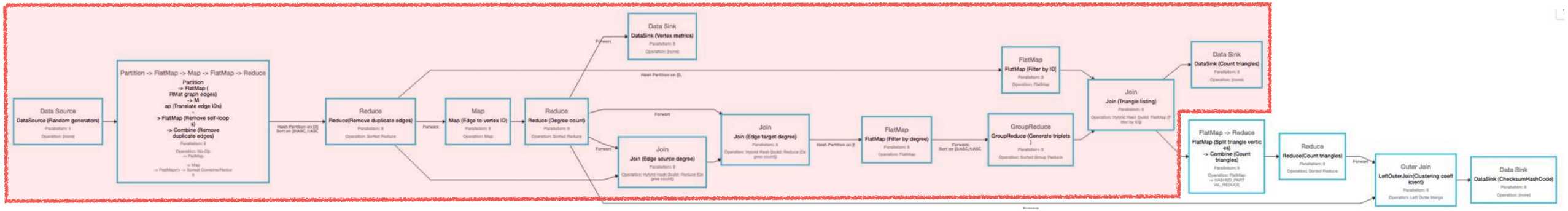
  instead look for triangles at the vertex with lowest degree

Implicit Object Reuse

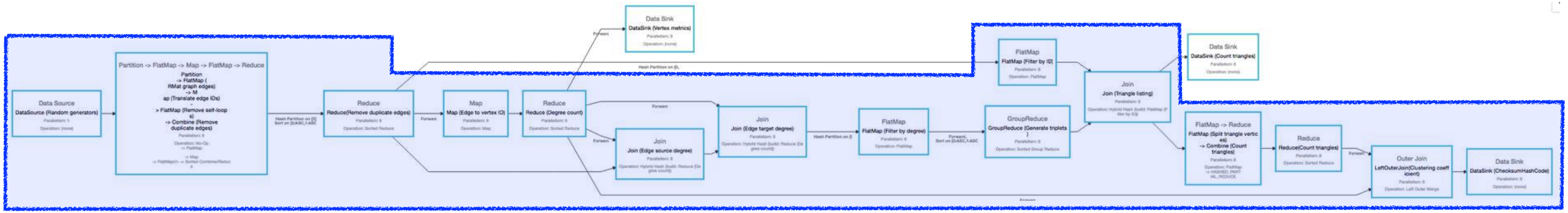


Implicit Operator Reuse

Global Clustering Coefficient



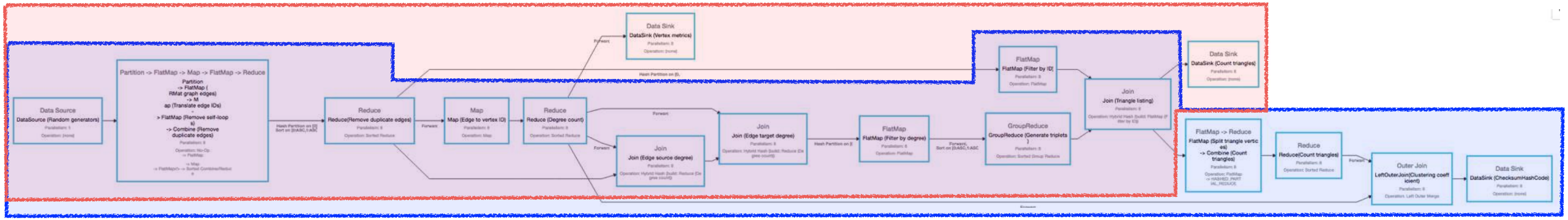
Implicit Operator Reuse



Local Clustering Coefficient

Implicit Operator Reuse

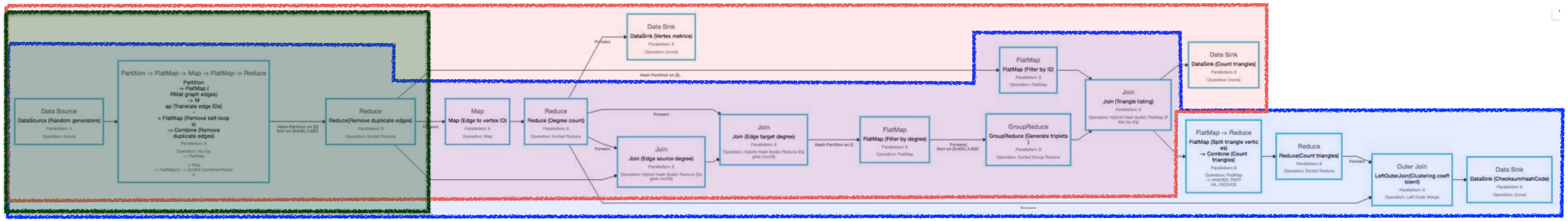
Global Clustering Coefficient



Local Clustering Coefficient

Implicit Operator Reuse

Global Clustering Coefficient



Graph

Local Clustering Coefficient

```
gcc = graph
   .run(new GlobalClusteringCoefficient<LongValue, NullValue, NullValue>());
lcc = graph
   .run(new LocalClusteringCoefficient<LongValue, NullValue, NullValue>());
```

Implicit Operator Reuse

- Three generations of operator reuse
  1. explicit reuse via get/set on intermediate DataSets
  2. Delegate using code generation
  3. NoOpOperator as placeholder in Flink 1.2
- Algorithms extend GraphAlgorithmWrappingDataSet/Graph

  and define and implement a mergeable configuration

GraphAnalytic

- Terminal; results retrieved via accumulators
- Defers execution to the user to allow composing multiple analytics and algorithms into a single program

```
public interface GraphAnalytic<K, VV, EV, T> {

  T getResult();

  T execute() throws Exception;

  T execute(String jobName) throws Exception;

  GraphAnalytic<K, VV, EV, T> run(Graph<K, VV, EV> input);
}
```

Similarity Measures

- Common Neighbors

$$CN_{A,B} = |A \cap B|$$

- Jaccard Index is common neighbors / distinct neighbors

$$JI_{A,B} = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

- Adamic-Adar is sum over weighted-degree of common neighbors

$$AA_{A,B} = \sum_{n:CN} log(\frac{1}{d_n})$$

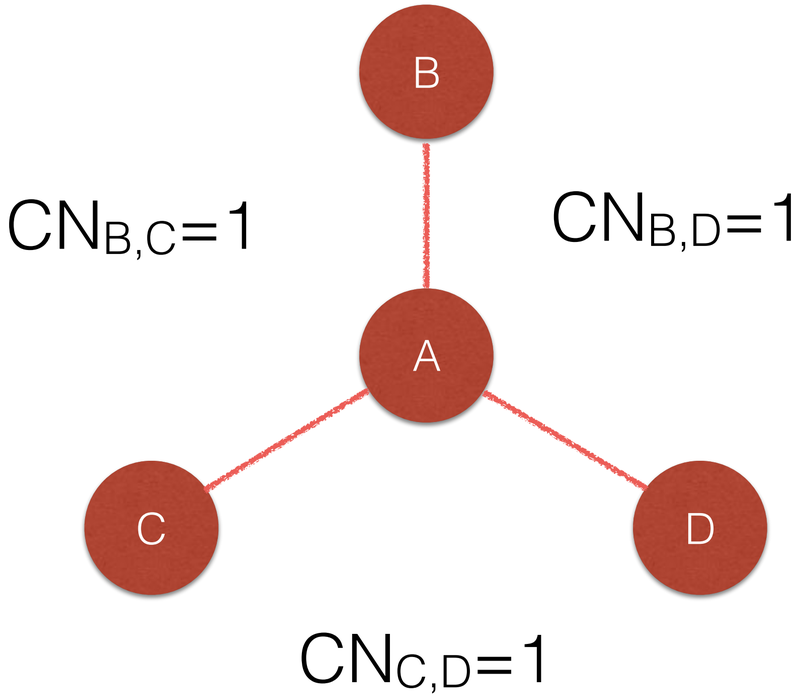
Common Neighbors

graph.getEdges()

.groupBy(0).crossGroup(...)

.groupBy(0, 1).sum(2)

GroupCross Input	GroupCross Input
<A,B>, <A,C>, <A,D>	<B,C>, <B,D>, <C,D>
<B,A>	
<C,A>	
<D,A>	

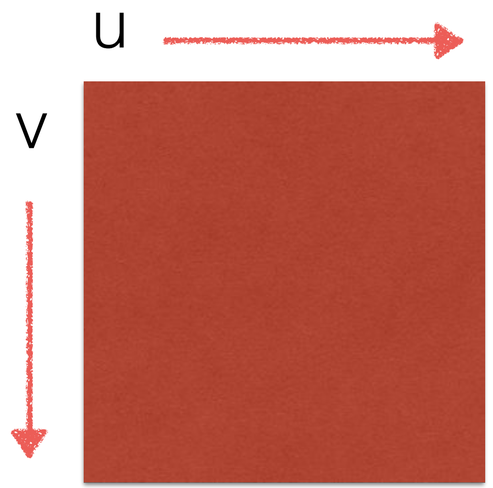


GroupedCross

- FLINK-1267: pair-wise compare or combine all elements of a group; currently can

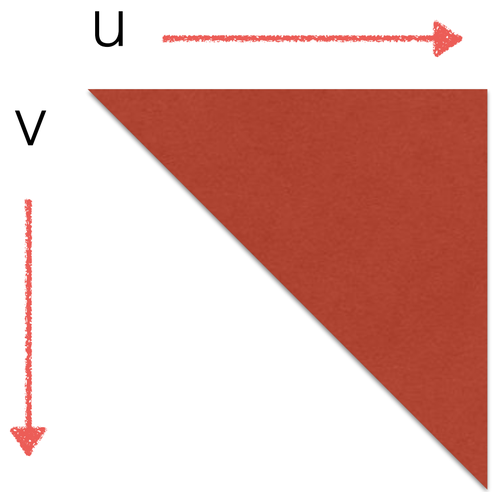
  groupReduce: consume and store the complete iterator in memory and build all pairs

  self-Join: the engine builds all pairs of the full symmetric Cartesian product.



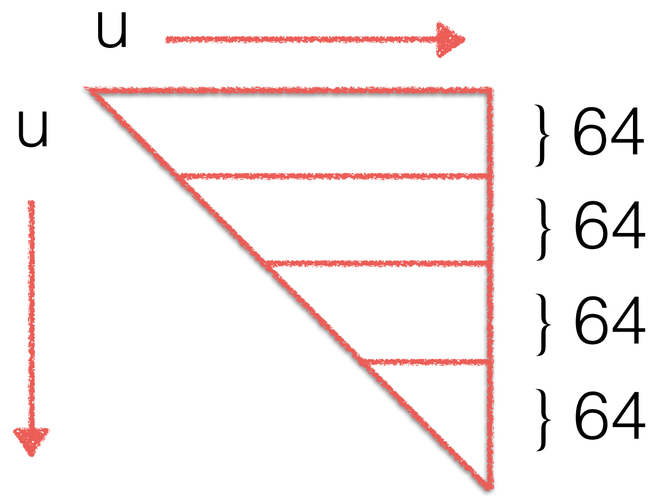
GroupedCross

- Hashes evenly partition linear outputs, but here quadratic



GroupedCross

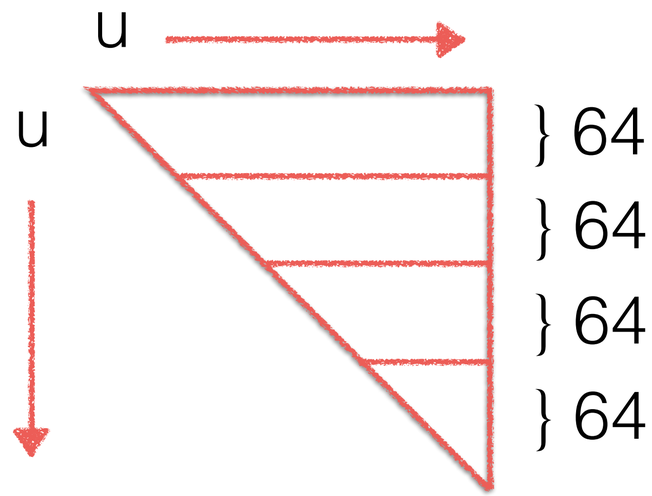
- Hashes evenly partition linear outputs, but here quadratic
- As linear operators:
  - process fixed width slices



GroupedCross

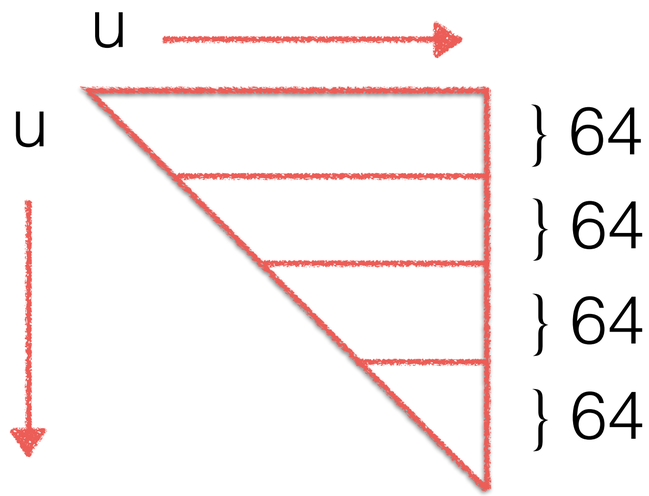
- Hashes evenly partition linear outputs, but here quadratic
- As linear operators:
  - process fixed width slices

    … generating slices is still quadratic!



GroupedCross

- Hashes evenly partition linear outputs, but here quadratic
- As linear operators:

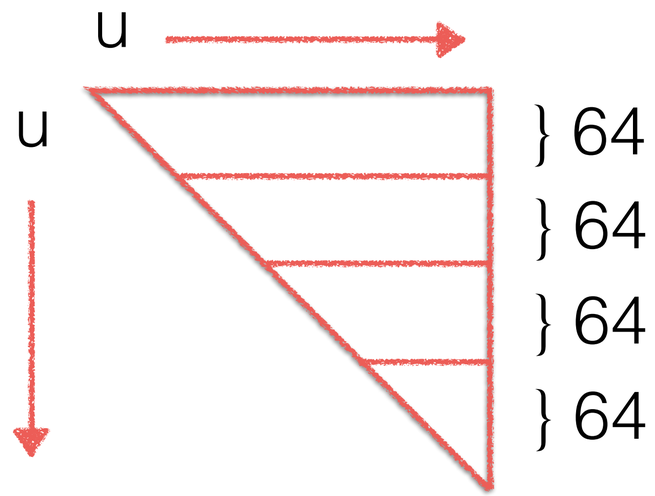


3. process fixed width slices

GroupedCross

- Hashes evenly partition linear outputs, but here quadratic
- As linear operators:

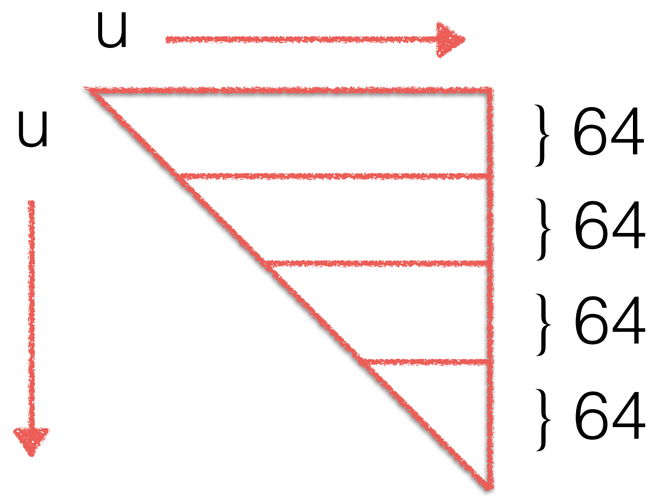
1. emit # slices for each element



3. process fixed width slices

GroupedCross

- Hashes evenly partition linear outputs, but here quadratic
- As linear operators:
  1. emit # slices for each element
  2. emit 1..# slice for each element
  3. process fixed width slices



Jaccard Index / Adamic-Adar

- Jaccard Index edges annotated with target degree: **<s, t, d(t)>**
- Adamic-Adar edges annotated with inverse logarithm of source degree: **<s, t, 1/log(d(s))>**
- groupBy, crossGroup, groupBy, sum scores as Common Neighbors

Flink Serialization

- “Think like an element”

- Flink aggressively serializes even when data fits in memory

MapDriver:

```
while (this.running && ((record = input.next(record)) != null)) {
    numRecordsIn.inc();
    output.collect(function.map(record));
}
```

- Complex types are antithetical to performance

  don’t stream ArrayList, HashSet, …

  do use Flink Value types and ValueArray types (FLINK-3695)

Modern Memory Model

- Processor registers — the fastest possible access (usually 1 CPU cycle). A few thousand bytes in size
- Cache
  - Level 0 (L0) Micro operations cache — 6 KiB in size
  - Level 1 (L1) Instruction cache — 128 KiB in size
  - Level 1 (L1) Data cache — 128 KiB in size. Best access speed is around 700 GiB/second
  - Level 2 (L2) Instruction and data (shared) — 1 MiB in size. Best access speed is around 200 GiB/second
  - Level 3 (L3) Shared cache — 6 MiB in size. Best access speed is around 100 GB/second
  - Level 4 (L4) Shared cache — 128 MiB in size. Best access speed is around 40 GB/second
- **Main memory** (Primary storage) — Gigabytes in size. Best access speed is around 10 GB/second.
- Disk storage (Secondary storage) — Terabytes in size. As of 2013, best access speed is from a solid state drive is about 600 MB/second
- Nearline storage (Tertiary storage) — Up to exabytes in size. As of 2013, best access speed is about 160 MB/second
- Offline storage

https://en.wikipedia.org/wiki/Memory_hierarchy

Object Reuse

- Performance boost when serializing/deserializing simple types

```
// Set up the execution environment
final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
env.getConfig().enableObjectReuse();
```

- Inputs may be modified between function calls;

  if storing inputs must make a copy

- Output may be modified pre-serialization by chained operators

- Configured globally; open discussion for using annotations

CopyableValue

- Implemented by Flink's value types (LongValue, StringValue, …)

```
public interface CopyableValue<T> extends Value {

  int getBinaryLength();

  void copyTo(T target);

  T copy();

  void copy(DataInputView source, DataOutputView target) throws IOException;
}
```

- Create new or copy to list of cached objects

```
if (visitedCount == visited.size()) {
  visited.add(edge.f1.copy());
} else {
  edge.f1.copyTo(visited.get(visitedCount));
}
```

Hints

- CombineHint.[HASH|SORT|NONE]
- JoinHint.[BROADCAST/REPARTITION_HASH_FIRST/SECOND
  |REPARTITION_SORT_MERGE]
- @ForwardedFields[|First|Second]

Agenda

- [x] Graphs with Gelly
- [x] Native algorithms
- [x] Configuring at scale
- [ ] Sort benchmark
- [ ] Profiling and benchmarking
- [ ] In development

Amazon Web Services

- c4.8xlarge:
  - 36 vcores
  - 60 GiB memory
  - 800 MB/s EBS
  - 10 Gbps networking + 4 Gbps dedicated EBS
- 2 NUMA cores (FLINK-3163)
- 5 x 200 GiB EBS volumes; kernel config: xen_blkfront.max=256
- Hourly Spot pricing ~$0.30 - $0.36 plus $0.14/hr EBS

Memory Allocation

- User defined functions
- TaskManager managed memory
- Network buffers
- Readhead buffers for spilled files in merge-sort

My flink-conf.yaml

jobmanager.rpc.address: localhost
jobmanager.rpc.port: 6123
jobmanager.heap.mb: 1024
taskmanager.heap.mb: 18000
taskmanager.numberOfTaskSlots: 18
taskmanager.memory.preallocate: true
parallelism.default: 36
jobmanager.web.port: 8081
webclient.port: 8080
taskmanager.network.numberOfBuffers: 25920
taskmanager.tmp.dirs: /volumes/xvdb/tmp:/volumes/xvdc/tmp:/volumes/xvdd/tmp:...

akka.ask.timeout: 1000 s
akka.lookup.timeout: 100 s
jobmanager.web.history: 25
taskmanager.memory.fraction: 0.9
taskmanager.memory.off-heap: true
taskmanager.runtime.hashjoin-bloom-filters: true
taskmanager.runtime.max-fan: 8192

Configuring Network Buffers

- Ignore heuristic as recommended in documentation

  “#slots-per-TM^2 * #TMs * 4”

- Instead introspect with statsd reporter

  nc -l -u -k 0.0.0.0 <port> | grep “AvailableMemorySegments”

  AvailableMemorySegments, TotalMemorySegments in Flink 1.2

- May show up in the web UI

Python statsd server

```
#!/usr/bin/env python

import socket
import sys
import time

UDP_IP = ''
UDP_PORT = int(sys.argv[1])

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

while True:
    data, addr = sock.recvfrom(4096)
    print('{:.6f} {}'.format(time.time(), data))
```

“Little” Parallelism

- Number of buffers per TM increases linearly with size of cluster so can run out of buffers or consume too much memory

  18 * (36 * 64) * 32 KiB = 1.36 GB for a single shuffle

- Typically have small tasks early and late and large tasks in middle
- Flink runs slowly when parallelism mismatched to size of data
- Must be paired with a scheduler which evenly allocates slots among TaskManagers

Agenda

- [x] Graphs with Gelly
- [x] Native algorithms
- [x] Configuring at scale
- [x] Sort benchmark
- [ ] Profiling and benchmarking
- [ ] In development

Cloudsort

- sortbenchmark.org: lowest cost to sort 100 TB in a public cloud

  an “I/O” benchmark, no compression permitted

- Input and output on persistent storage

  replicated storage outside the cluster

- No discounts!

- Amazon has better compute, Google has better networking

SchnellSort

- Everything at https://github.com/greghogan/flink-cloudsort
- Custom IndyRecord type
- 1 x c4.4xlarge master, 128 x c4.4xlarge workers
- 3 x 255 GiB EBS for tmp directories
- 1 GB input files in S3
- 256 MB output files in S3

Sort Cost

	Price	Run 1	Run 2	Run 3
Time		7133 s	6561 s	6706 s
AWS c4.4xlarge Instances		129	129	129
AWS c4.4xlarge Cost	$0.838/instance-hr	$214.20	$197.02	$201.38
AWS EBS gp2 GiB		98994	98994	98994
AWS EBS gp2 Cost	$0.10/GiB-mo	$27.25	$25.06	$25.62
AWS S3 Cost	$0.024/GiB-mo	$7.59	$7.09	$7.23
AWS S3 LIST, PUT		401,537	401,627	401,541
AWS S3 LIST, PUT Cost	$0.005 per 1,000	$2.01	$2.01	$2.01
AWS S3 GET		100,330	100,149	100,185
AWS S3 GET Cost	$0.004 per 10,000	$0.05	$0.05	$0.05
Total Cost		$251.10	$231.23	$236.39

Java SSL Performance

- Cannot choose SSLEngine for HDFS stack

  solved with intrinsics in jdk8-b112 and Skylake’s Intel SHA Extensions

- Galois/Counter Mode (GCM) can be disabled

  https://docs.oracle.com/javase/8/docs/technotes/guides/security/PolicyFiles.html

- SHA-1 or MD5 are integral to SSL and cannot be disabled

  compare with `openssl speed md5 sha`

  used AWS CLI

Netty Configuration

- Linux TCP/IP default buffers

```
$ cat /proc/sys/net/ipv4/tcp_[rw]mem
4096  87380    3813280
4096  20480    3813280
```

- Netty configuration in flink-conf.yaml

```
taskmanager.net.num-arenas: 16
taskmanager.net.server.numThreads: 16
taskmanager.net.client.numThreads: 16
taskmanager.net.sendReceiveBufferSize: 67108864
```

Slow uploads

- “Counting Triangles and the Curse of the Last Reducer”

  kill and retry slow downloads and uploads

  write files to /dev/shm and upload using “thread-pool”

- Configurable timeout

  120 seconds for input

  60 seconds for output

Agenda

- [x] Graphs with Gelly
- [x] Native algorithms
- [x] Configuring at scale
- [x] Sort benchmark
- [x] Profiling and benchmarking
- [ ] In development

Java Flight Recorder

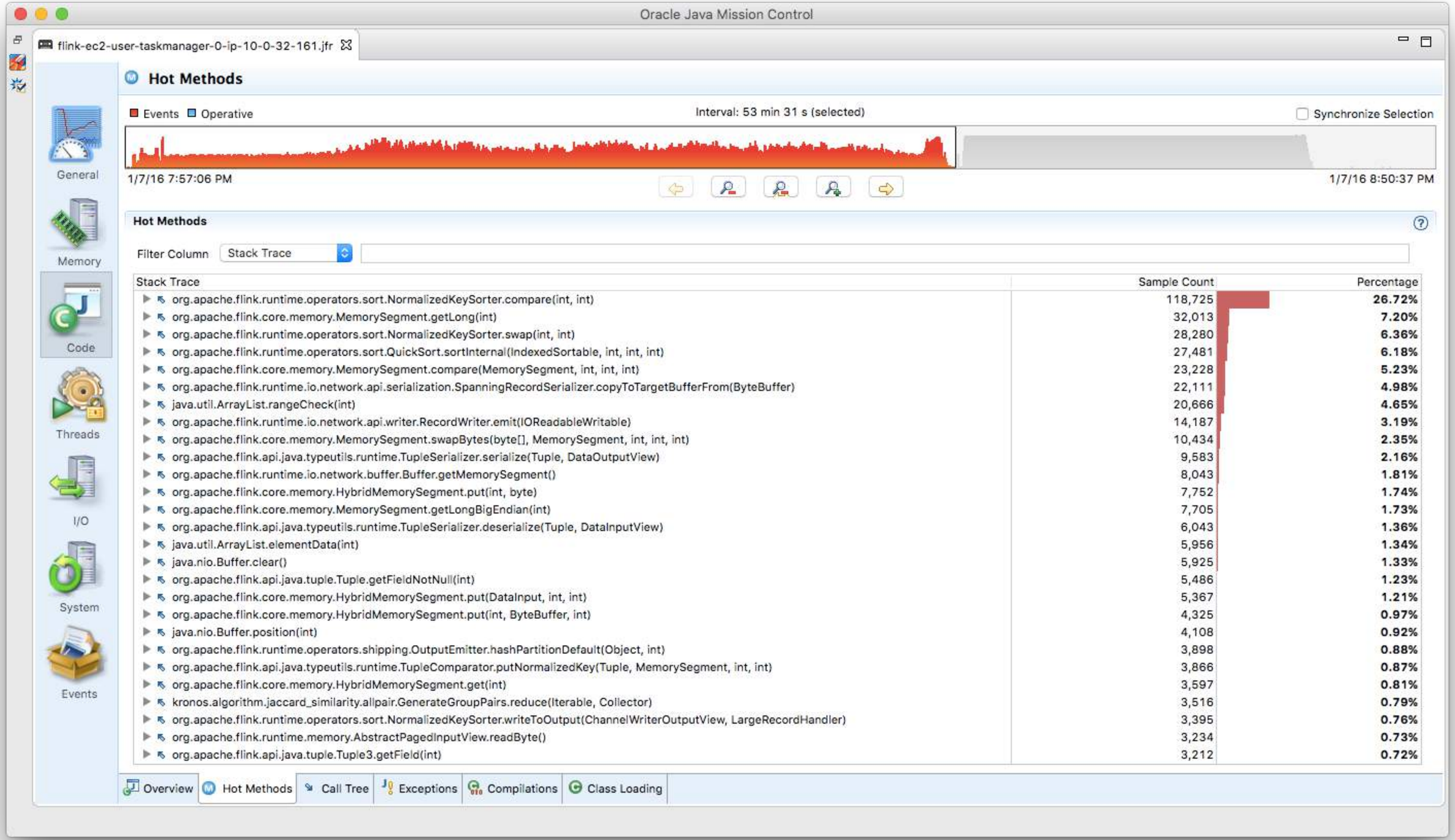
- Must edit flink-daemon.sh in order to rotate log file

```
jfr="${FLINK_LOG_DIR}/flink-${FLINK_IDENT_STRING}-${DAEMON}-${id}-${HOSTNAME}.jfr"
JFR_ARGS="-XX:+UnlockCommercialFeatures -XX:+FlightRecorder \
-XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints \
-XX:FlightRecorderOptions=defaultrecording=true,maxage=0,disk=true,dumponexit=true,dumponexitpath=${jfr}"
  ...
rotateLogFile $jfr
  ...
$JAVA_RUN $JFR_ARGS ...
```

- Can concatenate files from multiple TaskManagers

```
java oracle.jrockit.jfr.tools.ConCatRepository . -o <filename>.jfr
```

- Open jfr output files in JDK-provided **jmc**


Oracle Java Mission Control
flink-ec2-user-taskmanager-0-ip-10-0-32-161.jfr
Hot Methods
General
Memory
Code
Threads
I/O
System
Events
Events
Operative
Interval: 53 min 31 s (selected)
Synchronize Selection
1/7/16 7:57:06 PM
1/7/16 8:50:37 PM
Hot Methods
Filter Column
Stack Trace
Stack Trace
Sample Count
Percentage
org.apache.flink.runtime.operators.sort.NormalizedKeySorter.compare(int, int) 118,725 26.72%
org.apache.flink.core.memory.MemorySegment.getLong(int) 32,013 7.20%
org.apache.flink.runtime.operators.sort.NormalizedKeySorter.swap(int, int) 28,280 6.36%
org.apache.flink.runtime.operators.sort.QuickSort.sortInternal(IndexedSortable, int, int, int) 27,481 6.18%
org.apache.flink.core.memory.MemorySegment.compare(MemorySegment, int, int, int) 23,228 5.23%
org.apache.flink.runtime.io.network.api.serialization.SpanningRecordSerializer.copyToTargetBufferFrom(ByteBuffer) 22,111 4.98%
java.util.ArrayList.rangeCheck(int) 20,666 4.65%
org.apache.flink.runtime.io.network.api.writer.RecordWriter.emit(IOReadableWritable) 14,187 3.19%
org.apache.flink.core.memory.MemorySegment.swapBytes(byte[], MemorySegment, int, int, int) 10,434 2.35%
org.apache.flink.api.java.typeutils.runtime.TupleSerializer.serialize(Tuple, DataOutputView) 9,583 2.16%
org.apache.flink.runtime.io.network.buffer.Buffer.getMemorySegment() 8,043 1.81%
org.apache.flink.core.memory.HybridMemorySegment.put(int, byte) 7,752 1.74%
org.apache.flink.core.memory.MemorySegment.getLongBigEndian(int) 7,705 1.73%
org.apache.flink.api.java.typeutils.runtime.TupleSerializer.deserialize(Tuple, DataInputView) 6,043 1.36%
java.util.ArrayList.elementData(int) 5,956 1.34%
java.nio.Buffer.clear() 5,925 1.33%
org.apache.flink.api.java.tuple.Tuple.getFieldNotNull(int) 5,486 1.23%
org.apache.flink.core.memory.HybridMemorySegment.put(DataInput, int, int) 5,367 1.21%
org.apache.flink.core.memory.HybridMemorySegment.put(int, ByteBuffer, int) 4,325 0.97%
java.nio.Buffer.position(int) 4,108 0.92%
org.apache.flink.runtime.operators.shipping.OutputEmitter.hashPartitionDefault(Object, int) 3,898 0.88%
org.apache.flink.api.java.typeutils.runtime.TupleComparator.putNormalizedKey(Tuple, MemorySegment, int, int) 3,866 0.87%
org.apache.flink.core.memory.HybridMemorySegment.get(int) 3,597 0.81%
kronos.algorithm.jaccard_similarity.allpair.GenerateGroupPairs.reduce(Iterable, Collector) 3,516 0.79%
org.apache.flink.runtime.operators.sort.NormalizedKeySorter.writeToOutput(ChannelWriterOutputView, LargeRecordHandler) 3,395 0.76%
org.apache.flink.runtime.memory.AbstractPagedInputView.readByte() 3,234 0.73%
org.apache.flink.api.java.tuple.Tuple3.getField(int) 3,212 0.72%
Overview
Hot Methods
Call Tree
Exceptions
Compilations
Class Loading

Macro Benchmarks

- Performance test new features and stress test Flink releases

  https://github.com/greghogan/flink-benchmark

  multiple time samples on increasing scale; time-weighted

  {“algorithm”:”AdamicAdar","idType":"STRING","scale":12,"runtime_ms":634,"accumulators":{...}
  {“algorithm":"TriangleListingUndirected","idType":"INT","scale":16,"runtime_ms":813,"accumulator
  {“algorithm":"AdamicAdar","idType":"LONG","scale":12,"runtime_ms":410,"accumulators":{...}
  {“algorithm":"HITS","idType":"STRING","scale":16,"runtime_ms":1693,"accumulators":{...}
  {“algorithm”:”JaccardIndex”,”idType":"LONG","scale":12,"runtime_ms":431,"accumulators":{...}

- Profiling drives benchmarking: chasing the “1%”

- What metrics can be collected and analyzed?

Agenda

- Graphs with Gelly
- Native algorithms
- Configuring at scale
- Sort benchmark
- Profiling and benchmarking
- In development

Custom Types

- FLINK-3042: Define a way to let types create their own TypeInformation
- @TypeInfo

  -> TypeInfoFactory

  -> TypeInformation

  -> serializer/comparator and more
- Efficient serialization of arrays in FLINK-3695: ValueArray types

ValueArray<T>

```
@TypeInfo(ValueArrayTypeInfoFactory.class)
public interface ValueArray<T> ...


public class ValueArrayTypeInfoFactory<T> extends
TypeInfoFactory<ValueArray<T>> {

  @Override
  public TypeInformation<ValueArray<T>> createTypeInfo(...) {
    return new ValueArrayTypeInfo(genericParameters.get("T"));
  }
}
```

ValueArrayTypeInfo<T>

```
public class ValueArrayTypeInfo<T extends ValueArray> extends TypeInformation<T> implements AtomicType<T> {

    private final Class<T> typeClass;

    @Override
    @SuppressWarnings("unchecked")
    public TypeSerializer<T> createSerializer(ExecutionConfig executionConfig) {
        if (IntValue.class.isAssignableFrom(typeClass)) {
            return (TypeSerializer<T>) new CopyableValueSerializer(IntValueArray.class);
        } else...

    @SuppressWarnings({ "unchecked", "rawtypes" })
    @Override
    public TypeComparator<T> createComparator(boolean sortOrderAscending, ExecutionConfig executionConfig) {
        if (IntValue.class.isAssignableFrom(typeClass)) {
            return (TypeComparator<T>) new CopyableValueComparator(sortOrderAscending, IntValueArray.class);
        } else ...
```

ArrayableValue

- LongValueArray, StringValueArray, …
- Initial use cases:

  maximum value or top-K for pairwise similarity scores

  hybrid adjacency-list model:

u	v
u	w
u	x
y	z

Edge List

u	v, w, x
y	z

Adjacency List

u	v, w
u	x
y	z

Hybrid

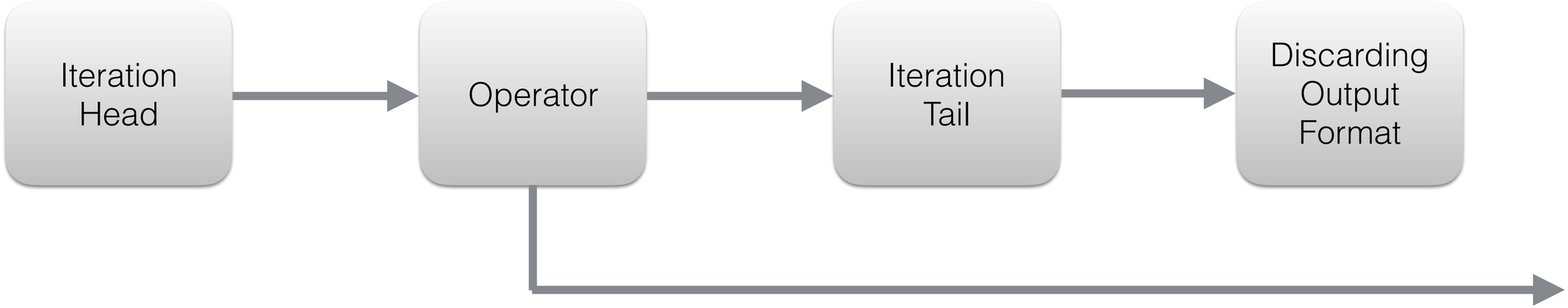
Iteration Intermediate Output

- For many iterative algorithms the output is ∅

  algorithm output is the output of intermediate operator

  breadth-first search, K-Core decomposition, K-Truss, …

- Delta iteration solution set is blocking, doesn’t scale



Coding to Interfaces

- Vertex/Edge as interface or abstract class

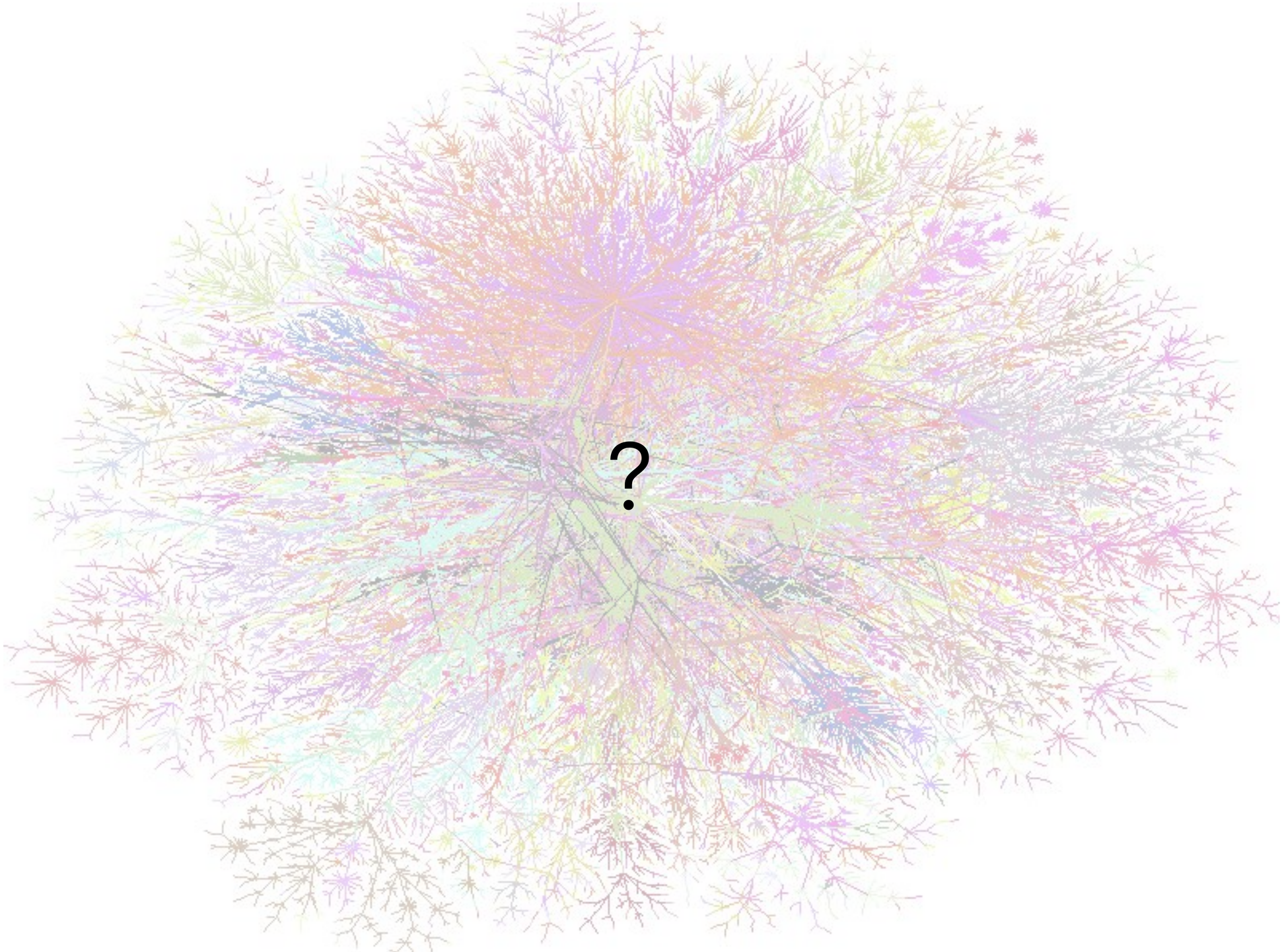
  subclasses define value fields

  eliminate “value packing”

- Requires code generation; must track fields from input to output
- Make algorithms truly modular
- Optimizer can prune unused fields

Why batch? Why Gelly?

- A framework for building great algorithms
- Scaling to 100s of algorithms
- As an out-of-the-box tool for graph analysis



Images

- Squirrel: https://flink.apache.org/img/logo/png/1000/flink_squirrel_1000.png
- Clouds: http://www.publicdomainpictures.net/pictures/120000/velka/large-cumulus-clouds-1.jpg
- Tree: https://motivatedbynature.com/wp-content/uploads/2016/04/tree_PNG2517.png
- RMat: http://www.cs.cmu.edu/~christos/PUBLICATIONS/siam04.pdf
- Triads: http://www.educa.fmf.uni-lj.si/datana/pub/networks/pajek/triade.pdf

Agenda Images

- Fernsehturm Berlin: http://readysettrek.com/wp-content/uploads/2015/09/berlin-attractions.jpg
- Reichstag: https://upload.wikimedia.org/wikipedia/commons/c/c7/Reichstag_building_Berlin_view_from_west_before_sunset.jpg
- Brandenburg Gate: https://images2.alphacoders.com/479/479507.jpg
- Schloss Charlottenburg : https://upload.wikimedia.org/wikipedia/commons/e/ec/Le_château_de_Charlottenburg_(Berlin)_(6340508573).jpg
- Memorial: https://s-media-cache-ak0.pinimg.com/736x/45/fa/85/45fa85018367cd88d4ffc0f62416c804.jpg
- Olympic Stadium: http://sumfinity.com/wp-content/uploads/2014/03/Olympic-Stadium-Berlin.jpg