Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications

By Ahmad and Cheung

Presented by: Ishank Jain

Department of Computer Science

03/19/2019

UNIVERSITY OF WATERLOO
FACULTY OF MATHEMATICS

CONTENT

- Background
- Research Question
- Method
- Results
- Conclusion
- Questions

BACKGROUND

- Implementations of MapReduce
- Source-to-Source Compilers
- Synthesizing Efficient Implementations
- Query Optimizers and IRs.

BACKGROUND: Implementations of MapReduce

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new

BACKGROUND: Source-to-Source Compilers

Translating Imperative Code to MapReduce

Cosmin Radoi
University of Illinois
cos@illinois.edu

Stephen J. Fink Rodric Rabbah
IBM T.J. Watson Research Center
{sjfink,rabbah}@us.ibm.com

Manu Sridharan
Samsung Research America
m.sridharan@samsung.com

Abstract

We present an approach for automatic translation of sequential, imperative code into a parallel MapReduce framework. Automating such a translation is challenging: imperative updates must be translated into a functional MapReduce form in a manner that both preserves semantics and enables parallelism. Our approach works by first translating the input code

stream MapReduce frameworks [1, 9] provide significant advantages for large-scale distributed parallel computation. In particular, MapReduce frameworks can transparently support fault-tolerance, elastic scaling, and integration with a distributed file system.

Additionally, MapReduce has attracted interest as a parallel programming model, independent of difficulties of distributed computation [24]. MapReduce has been shown to be

BACKGROUND: Synthesizing Efficient Implementations

MapReduce Program Synthesis

Calvin Smith
University of Wisconsin–Madison, USA

Aws Albarghouthi
University of Wisconsin–Madison, USA

Abstract

By abstracting away the complexity of distributed systems, large-scale data processing platforms—MapReduce, Hadoop, Spark, Dryad, etc.—have provided developers with simple means for harnessing the power of the cloud. In this paper, we ask whether we can *automatically synthesize* MapReduce-style distributed programs from input–output examples. Our ultimate goal is to enable end users to specify large-scale data analyses through the simple interface of examples. We thus present a new algorithm and tool for

complexity of distributed computing, e.g., node failures, load balancing, network topology, distributed protocols, etc.

By adding a layer of *abstraction* on top of distributed systems and providing developers with a restricted API, large-scale data processing platforms have become household names and indispensable tools for the modern software developer and data analyst. In this paper, we ask whether we can *raise* the level of abstraction even higher than what state-of-the-art platforms provide, but this time with the goal of unleashing the power of cloud computing for the average

BACKGROUND: Query Optimizers and IRs.

Tupleware: Redefining Modern Analytics

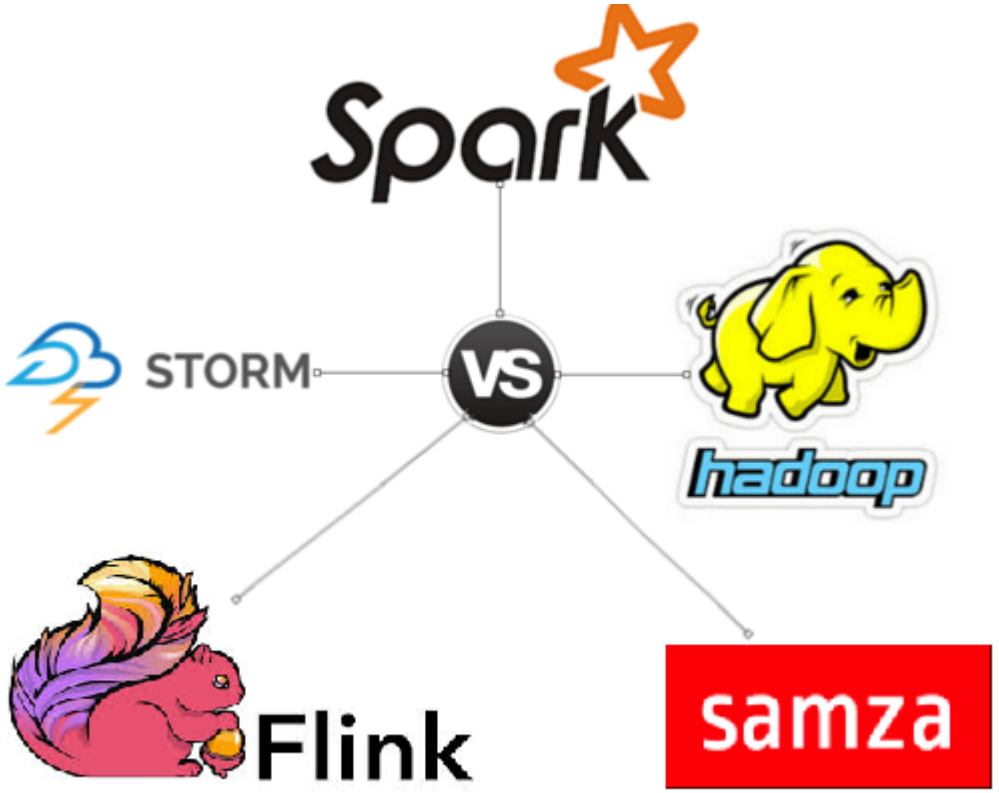
Andrew Crotty, Alex Galakatos, Kayhan Dursun, Tim Kraska, Ugur Cetintemel, Stan Zdonik
Department of Computer Science, Brown University
{crottyan, agg, kayhan, kraskat, ugur, sbz}@cs.brown.edu

Abstract

There is a fundamental discrepancy between the targeted and actual users of current analytics frameworks. Most systems are designed for the data and infrastructure of the Googles and Facebooks of the world—petabytes of data distributed across large cloud deployments consisting of thousands of cheap commodity machines. Yet, the vast majority of users operate clusters ranging from a few to a few dozen nodes, analyze relatively small datasets of up to several terabytes, and perform primarily compute-

Supporting the typical user, then, fundamentally changes the way we should design analytics tools. Current analytics frameworks are built around the major bottlenecks of large cloud deployments, in which data movement (disk to machine and across the network) is the primary performance bottleneck, machines are slow, and failures are the norm [19]. Conversely, with smaller clusters ranging in size from a few to a few dozen nodes, failures are the exception. Most importantly, whereas single-node performance is largely irrelevant in cloud deployments, it can no longer be ignored when targeting small clusters

MOTIVATION



CASPER

- **Casper** is a compiler that can **automatically retarget** sequential **Java programs** to Big Data processing frameworks such as **Spark, Hadoop or Flink**.

Image credit: https://casper.uwplse.org

CASPER

Access Path Selection
in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and retrieval. Nor does a user specify in what order joins are to be performed. The System R optimizer chooses both join order and an access path for each table in the SQL statement. Of the many possible choices, the optimizer chooses the one

MapReduce OPERATORS

- Map operator:
  - Converts a value of type τ into a multiset of key-value pairs of types κ and ν.
- Reduce operator:
  - Combines two values of type ν to produce a final value.
  - Shuffling.

$$map : (mset[\tau], \lambda_m) \longrightarrow mset[(\kappa, v)]$$
$$\lambda_m : \tau \longrightarrow mset[(\kappa, v)]$$

$$reduce : (mset[(\kappa, v)], \lambda_r) \longrightarrow mset[(\kappa, v)]$$
$$\lambda_r : (v, v) \longrightarrow v$$

PROGRAM SUMMARY

- The program summary, a high-level intermediate representation (IR), describes how the output of the code fragment (i.e., m) can be computed using a series of map and reduce stages from the input data (i.e., mat)

@Summary(

$$m = map(reduce(map(mat, \lambda_{m1}), \lambda_r), \lambda_{m2})$$

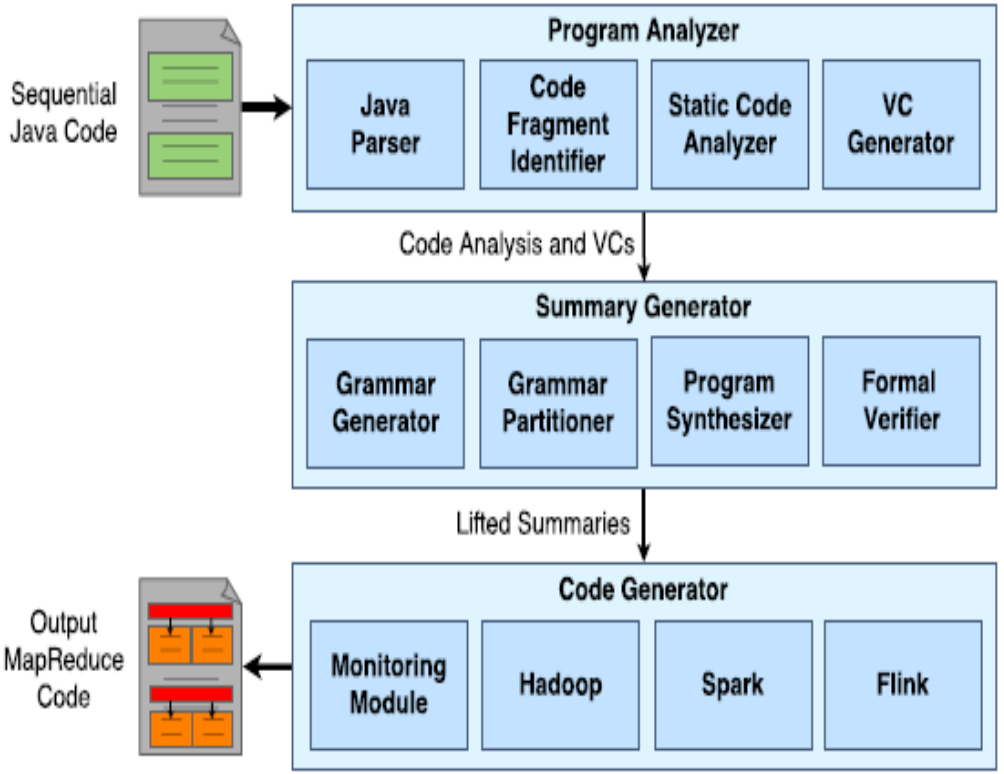
$$\lambda_{m1} : (i, j, v) \rightarrow \{(i, v)\}$$

$$\lambda_r : (v_1, v_2) \rightarrow v_1 + v_2$$

$$\lambda_{m2} : (k, v) \rightarrow \{(k, v/cols)\}$$)

SYSTEM ARCHITECTURE

- Program analyzer:
  - search space description
  - Verification condition
- Summary generator.
- Code generator.



PROGRAM SUMMARIES

- High level IR:
  - To express summaries that are translatable into the target API.
  - Let the synthesizer efficiently search for summaries that are equivalent to the input program.
- Limited number of operations.

$$
\begin{array}{rcl}
PS & := & \forall v.\ v = MR \mid \forall v.\ v = MR[v_{id}] \\
MR & := & map(MR, \lambda_m) \mid reduce(MR, \lambda_r) \mid join(MR, MR) \mid data \\
\lambda_m & := & f : (val) \rightarrow \{Emit\} \\
\lambda_r & := & f : (val_1, val_2) \rightarrow Expr \\
Emit & := & emit(Expr, Expr) \mid if(Expr)\ emit(Expr, Expr) \mid \\
 & & if(Expr)\ emit(Expr, Expr)\ else\ Emit \\
Expr & := & Expr\ op\ Expr \mid op\ Expr \mid f(Expr, Expr, \ldots) \mid \\
 & & n \mid var \mid (Expr, Expr)
\end{array}
$$

$$
\begin{array}{rclrcl}
v & \in & Output\ Variables & v_{id} & \in & Variable\ ID, \\
op & \in & Operators & f & \in & Library\ Methods
\end{array}
$$

SEARCH SPACE

- To generate the search space grammar, Casper analyzes the input.
- Code analyzer:
  - Dataflow analysis
  - Scanning function

$$
\begin{array}{rcl}
PS & := & \forall v.\ v = MR \mid \forall v.\ v = MR[v_{id}] \\
MR & := & map(MR, \lambda_m) \mid reduce(MR, \lambda_r) \mid join(MR, MR) \mid data \\
\lambda_m & := & f : (val) \rightarrow \{Emit\} \\
\lambda_r & := & f : (val_1, val_2) \rightarrow Expr \\
Emit & := & emit(Expr, Expr) \mid if(Expr)\ emit(Expr, Expr) \mid \\
 & & if(Expr)\ emit(Expr, Expr)\ else\ Emit \\
Expr & := & Expr\ op\ Expr \mid op\ Expr \mid f(Expr, Expr, \ldots) \mid \\
 & & n \mid var \mid (Expr, Expr)
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
v & \in & Output\ Variables & v_{id} & \in & Variable\ ID, \\
op & \in & Operators & f & \in & Library\ Methods
\end{array}
$$

SEARCH SPACE

Property	G_1	G_2	G_3
Map/Reduce Sequence	m	m → r	m → r → m
# Emits in λ_m	1	2	2
Key/Value Type	int	int	int or Tuple<int,int>

$$G1 \coloneqq map(mat, \lambda_m)$$

$$\lambda_m \coloneqq \begin{cases} (i,j,v) \to [(i,j)] \\ (i,j,v) \to [(i,v)] \\ (i,j,v) \to [(j,v+i)] \\ (i,j,v) \to [(i+j,v)] \\ \vdots \end{cases}$$

$$G2 \coloneqq reduce(map(mat, \lambda_m), \lambda_r)$$

$$\lambda_m \coloneqq \begin{cases} (i,j,v) \to [(i,v)] \\ (i,j,v) \to [(j,v+i)] \\ (i,j,v) \to [(i,j),(v,1)] \\ \vdots \end{cases}$$

$$\lambda_r \coloneqq \begin{cases} (v_1,v_2) \to v_1 \\ (v_1,v_2) \to v_2 + 4 \\ (v_1,v_2) \to v_1 + v_2 \\ \vdots \end{cases}$$

$$G3 \coloneqq map(reduce(map(mat, \lambda_{m1}), \lambda_r), \lambda_{m2})$$

$$\lambda_{m1} \coloneqq \begin{cases} (i,j,v) \to [(i,v)] \\ (i,j,v) \to [(i,(v,i))] \\ (i,j,v) \to [(i+1,j-v),(i,v)] \\ \vdots \end{cases}$$

$$\lambda_r \coloneqq \begin{cases} (v_1,v_2) \to v_1 \\ (v_1,v_2) \to v_1 + v_2 \\ (v_1,v_2) \to (v_1.1, v_2.2) \\ \vdots \end{cases}$$

$$\lambda_{m2} \coloneqq \begin{cases} (k,v) \to [(k,v),(v,k)] \\ (k,v) \to [((v.1,k),v.2)] \\ (k,v) \to [(k,v/cols)] \\ (k,v) \to if(v > i)[(k,v)] \\ \vdots \end{cases}$$

VERIFYING SUMMARIES

- Verification conditions:
  - Hoare logic
  - Predicate logic

$$invariant(m, i) \equiv \quad 0 \leq i \leq rows \ \wedge$$
$$m = map(reduce(map(mat[0..i], \lambda_{m1}), \lambda_r), \lambda_{m2})$$

(a) Outer loop invariant

Initiation	$(i = 0) \rightarrow Inv(m, i)$
Continuation	$Inv(m, i) \wedge (i < rows) \rightarrow$ $Inv(m[i \mapsto sum(mat[i])/cols], i + 1)$
Termination	$Inv(m, i) \wedge \neg(i < rows) \rightarrow PS(m, i)$

SEARCH STRATEGY

- Input:
  - a set of candidate summaries and invariants encoded as a grammar,
  - The correctness specification for the summary in the form of verification conditions.
- CEGIS Algorithm

```
function synthesize (G, VC):
  Φ = {} // set of random program states
  while true do
    ps, inv_{1..n} = generateCandidate(G, VC, Φ)
    if ps is null then return null // search space exhausted
    ϕ = boundedVerify(ps, inv_{1..n}, VC)
    if ϕ is null then return (ps, inv_{1..n}) // summary found
    else Φ = Φ ∪ ϕ // counter-example found

function findSummary (A, VC):
  G = generateGrammar(A)
  Γ = generateClasses(G)
  Ω = {} // summaries that failed verification
  Δ = {} // summaries that passed verification
  for γ ∈ Γ do
    while true do
      c = synthesize(γ - Ω - Δ, VC)
      if c is null and Δ is null then
        break // move to next grammar class
      else if c is null then
        return Δ // search complete
      else if fullVerify(c, VC) then Δ = Δ ∪ c
      else Ω = Ω ∪ c
  return null // no solution found
```

IMPROVISATION

- Verifier failures:
  - Casper must first prevent summaries that failed the theorem prover from being regenerated by the synthesizer.
- Incremental grammar generation:
  - Helps find summaries quicker and is more syntactically expressive.

```
function synthesize (G, VC):
  Φ = {} // set of random program states
  while true do
    ps, inv_{1..n} = generateCandidate(G, VC, Φ)
    if ps is null then return null // search space exhausted
    ϕ = boundedVerify(ps, inv_{1..n}, VC)
    if ϕ is null then return (ps, inv_{1..n}) // summary found
    else Φ = Φ ∪ ϕ // counter-example found

function findSummary (A, VC):
  G = generateGrammar(A)
  Γ = generateClasses(G)
  Ω = {} // summaries that failed verification
  Δ = {} // summaries that passed verification
  for γ ∈ Γ do
    while true do
      c = synthesize(γ - Ω - Δ, VC)
      if c is null and Δ is null then
        break // move to next grammar class
      else if c is null then
        return Δ // search complete
      else if fullVerify(c, VC) then Δ = Δ ∪ c
      else Ω = Ω ∪ c
  return null // no solution found
```

IMPROVISATION

- Search Algorithm for summaries:
  - Each synthesized summary (correct or not) is eliminated from the search space, forcing the synthesizer to generate a new summary each time.
  - When the grammar is exhausted, Casper returns the set of correct summaries Δ if it is non-empty

```
function synthesize (G, VC):
  Φ = {} // set of random program states
  while true do
    ps, inv_{1..n} = generateCandidate(G, VC, Φ)
    if ps is null then return null // search space exhausted
    ϕ = boundedVerify(ps, inv_{1..n}, VC)
    if ϕ is null then return (ps, inv_{1..n}) // summary found
    else Φ = Φ ∪ ϕ // counter-example found

function findSummary (A, VC):
  G = generateGrammar(A)
  Γ = generateClasses(G)
  Ω = {} // summaries that failed verification
  Δ = {} // summaries that passed verification
  for γ ∈ Γ do
    while true do
      c = synthesize(γ - Ω - Δ, VC)
      if c is null and Δ is null then
        break // move to next grammar class
      else if c is null then
        return Δ // search complete
      else if fullVerify(c, VC) then Δ = Δ ∪ c
      else Ω = Ω ∪ c
  return null // no solution found
```

COST MODEL

- Dynamic cost estimation:
  - It counts the number of unique data values that are emitted as keys.

$$cost_m(\lambda_m, N, W_m) = W_m * N * \sum_{i=1}^{|\lambda_m|} sizeOf(emit_i) * p_i$$

$$cost_r(\lambda_r, N, W_r) = W_r * N * sizeOf(\lambda_r) * \epsilon(\lambda_r)$$

$$cost_j(N_1, N_2, W_j) = W_j * N_1 * N_2 * sizeOf(emit_j) * p_j$$

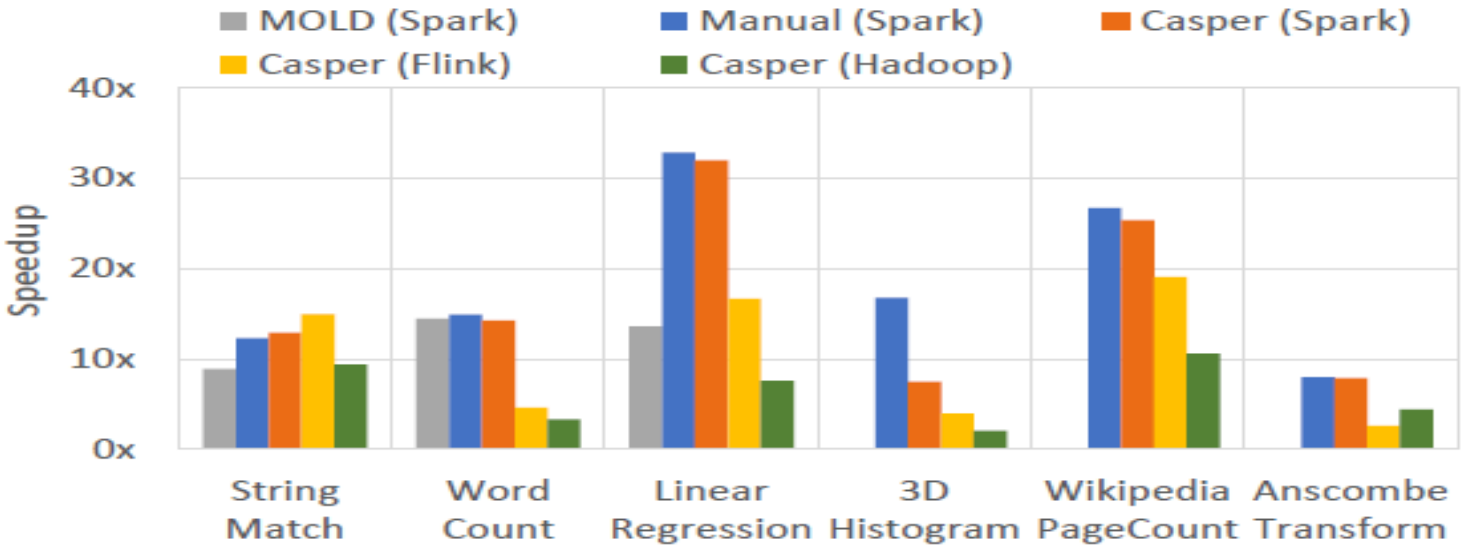
IMPORTANT POINTS AND LIMITATION

- The IR does not currently model the full range of operators across different MapReduce implementations.
- Biasing the search towards smaller grammars likely produces program summaries that run more efficiently. Although this is not sufficient to guarantee optimality of generated summaries. It's a tradeoff between efficient solution and time spent to generate the grammar.
- Casper can currently do this for basic Java statements, conditionals, functions, user-defined types, and loops.
- Recursive methods and methods with side-effects are not currently supported.

EVALUATION

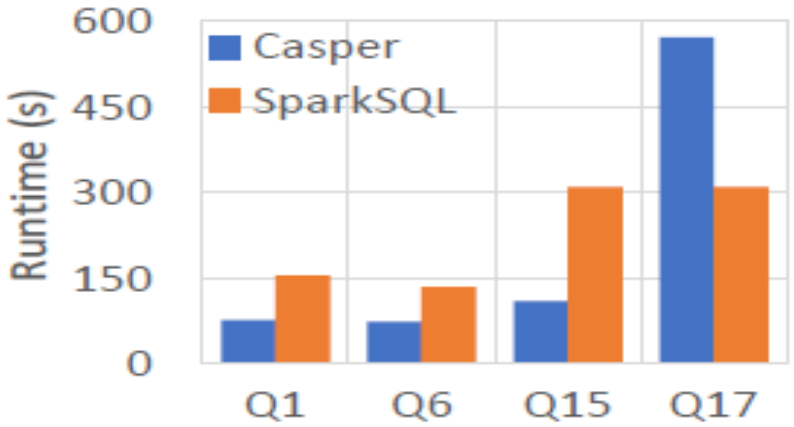
Suite	# Translated	Mean Speedup	Max Speedup
Phoenix	7 / 11	14.8x	32x
Ariths	11 / 11	12.6x	18.1x
Stats	18 / 19	18.2x	28.9x
Bigλ	6 / 8	21.5x	32.2x
Fiji	23 / 35	18.1x	24.3x
TPC-H	10 / 10	31.8x	48.2x
Iterative	7 / 7	18.4x	28.8x

EVALUATION

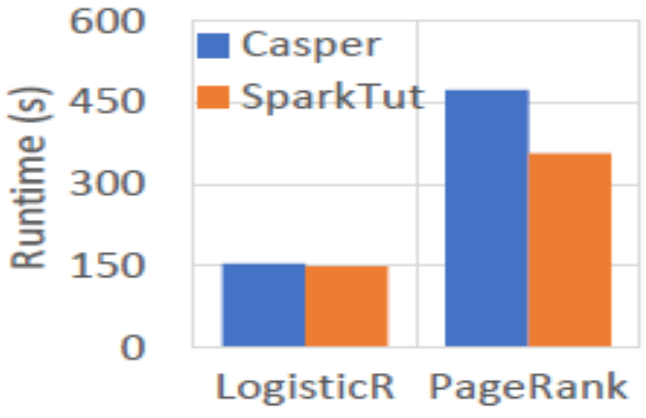


(a) CASPER achieves speedup competitive with manual translations

EVALUATION



(b) TPC-H benchmarks



(c) Iterative algorithms

EVALUATION

Source	Mean Time (s)	Mean LOC	Mean # Op	Mean TP Failures
Phoenix	944	13.8 (13.1)	2.3 (2.1)	0.35
Ariths	223	9.4 (7.6)	1.6 (1.2)	4
Stats	351	7.6 (5.8)	1.8 (1.8)	0.6
Bigλ	112	13.6 (10)	1.8 (2.0)	0.4
Fiji	1294	7.2 (7.4)	1.4 (1.6)	0.1
TPC-H	476	5.9 (n/a)	7.25 (n/a)	0
Iterative	788	3.3 (3.7)	4.5 (3.5)	2

EVALUATION

Benchmark	With Incr. Grammar	Without Incr. Grammar
WordCount	2	827
StringMatch	24	416
Linear Regression	1	94
3D Histogram	5	118
YelpKids	1	286
Wikipedia PageCount	1	568
Covariance	5	11
Hadamard Product	1	484
Database Select	1	397
Anscombe Transform	2	78

QUESTIONS

- Casper covers limited set of operations and doesn't perform well on ML related and Scientific images dataset. Does this make it usable only for beginner programmers?
- "Summaries are restricted to only those expressible using the IR, which lacks many features (e.g., pointers) that a general purpose language would have". Does this restrict the scope of finding a better target code?
- Certain methods such as recursive methods are not supported(reason: they don't gain any speedup). Is the paper not addressing issues that are essential part of general purpose coding?
  - NOTE: The paper wanted to reduce complexity for user to learn multiple DSL.

REFERENCE

Maaz Bin Safeer Ahmad, Alvin Cheung. Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications. *Proc. ACM SIGMOD International Conference on Management of Data*, pages 1205-1220, 2018.