Towards Transparent Combination of Model Management Execution Strategies for Low-Code Development Platforms

Jolan Philippe[12], Héléne Coullon[123], Massimo Tisi[12], and Gerson Sunyé[24]
IMT Atlantique[1] , LS2N[2], Inria[3], Université de Nantes[4]

October 20th, 2020

23rd International Conference on
Model-Driven Engineering Languages and Systems
MODELS 20

Introduction

LCDP manipulates model:

- At design time
- At runtime

Introduction

LCDP manipulates model:

- At design time
- At runtime

Large models? Modification frequency? Connectivity?

NEED OF ADAPTED TECHNIQUES

Introduction

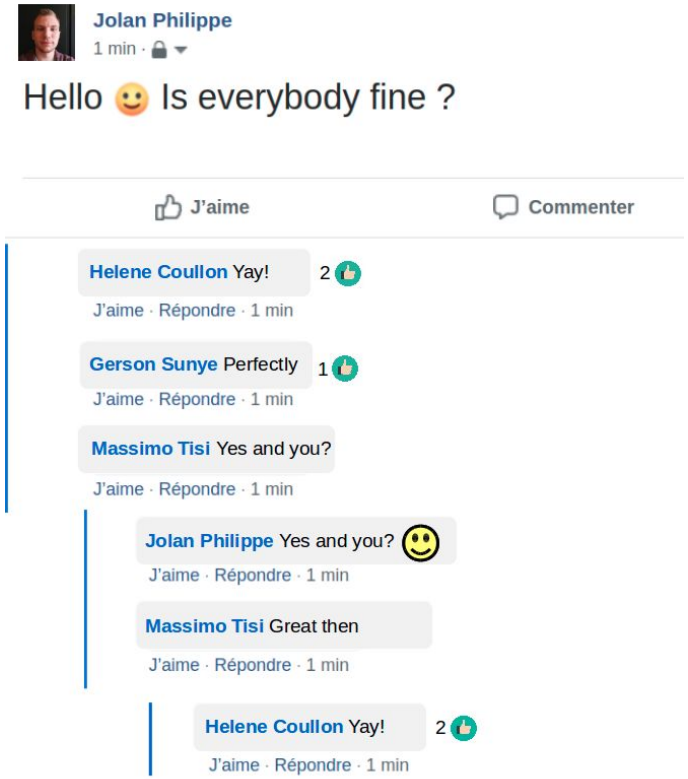
LCDP manipulates model:
- At design time
- At runtime

Large models? Modification frequency? Connectivity?

NEED OF ADAPTED TECHNIQUES

Running example: Build a LCDP to manipulate social networks
- Large datasets (huge social graph)
- Often modified
- Large number of users

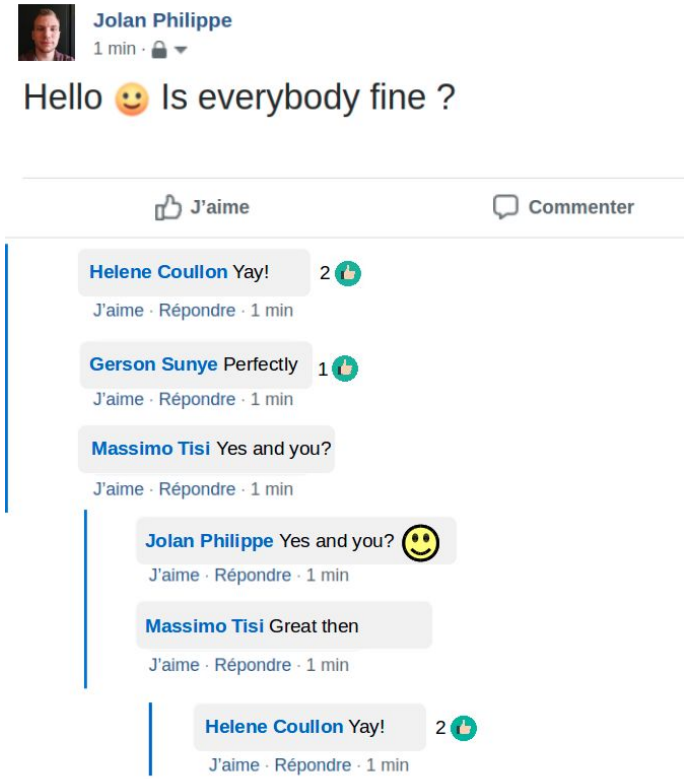
Transformation Tool Contest 2018



- Query:

  What is the most debated post in a social network?

Transformation Tool Contest 2018

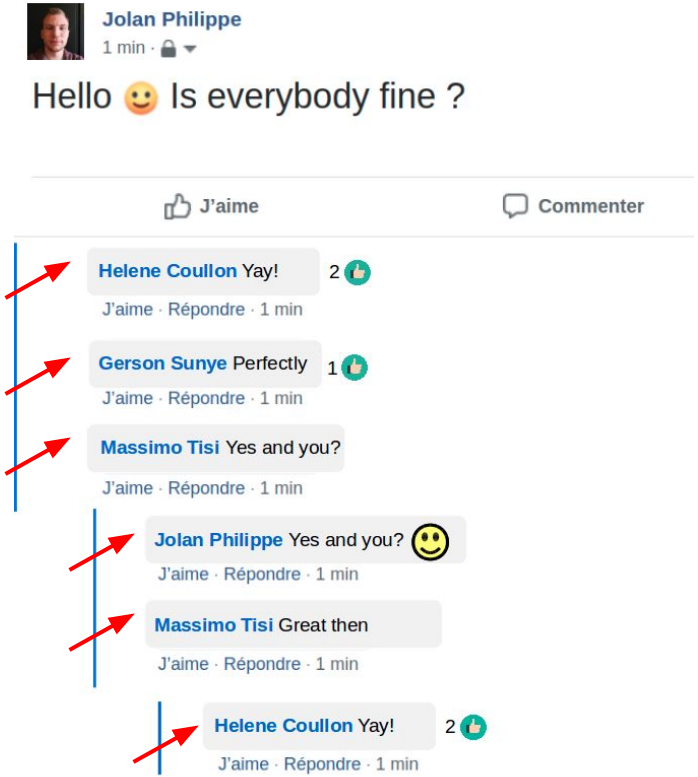


- Query:

  What is the most debated post in a social network?

  Score(post) = 10 * #(post.comments) + #(post.comments.likes)

Transformation Tool Contest 2018



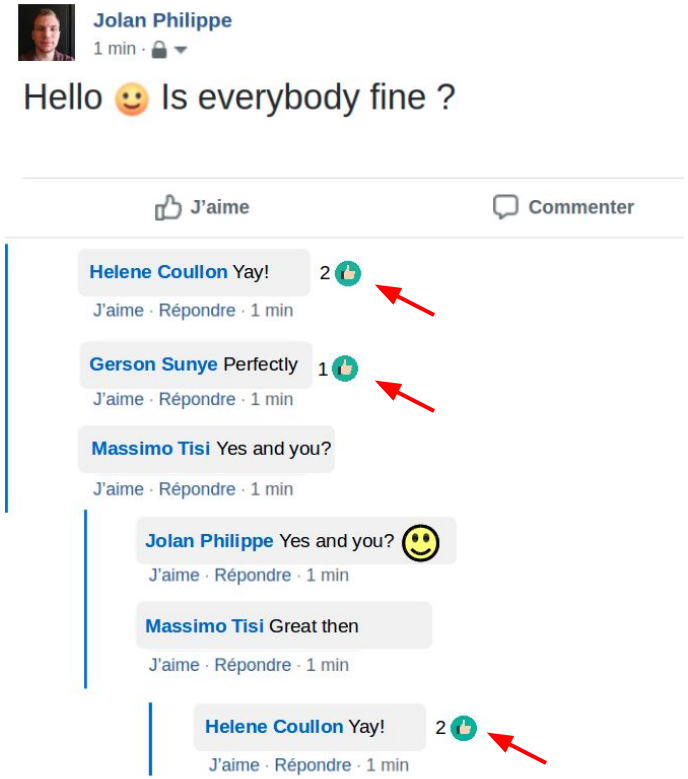
- Query:

  What is the most debated post in a social network?

  Score(post) = 10 * #(post.comments) + #(post.comments.likes)

- **#(post.comments) = 6**

Transformation Tool Contest 2018

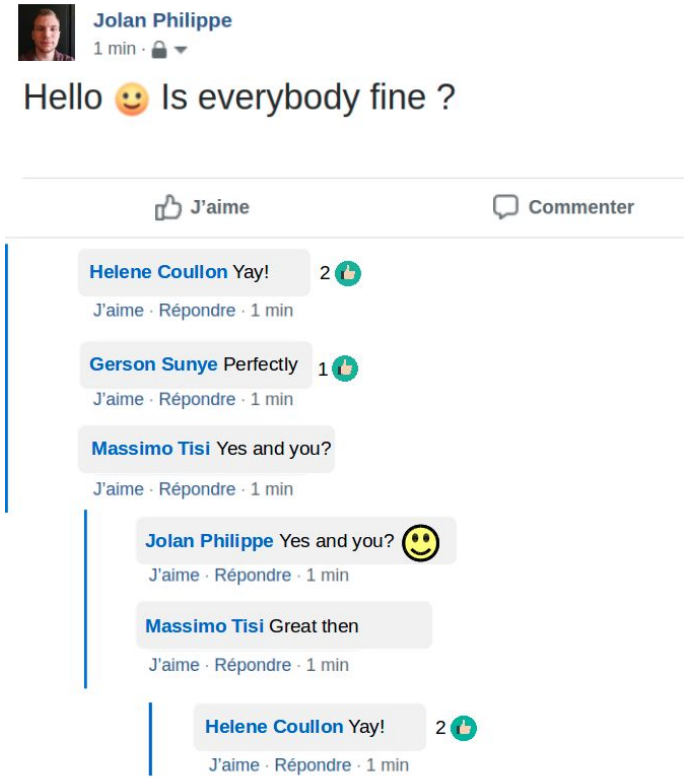


- Query:

  What is the most debated post in a social network?

  Score(post) = 10 * #(post.comments) + #(post.comments.likes)
- **#(post.comments) = 6**
- **#(post.comments.likes) = 5**

Transformation Tool Contest 2018



- Query:

  What is the most debated post in a social network?

  Score(post) = 10 * #(post.comments) + #(post.comments.likes)
- **#(post.comments) = 6**
- **#(post.comments.likes) = 5**
- **Score(post) = 65**

Many execution strategies

- **Reactivity**
  - Incrementality
  - Laziness
- **Parallelism**
  - Data-Parallelism
  - Task-Parallelism
  - Asynchronism

Many execution strategies

- **Reactivity**
  - Incrementality
  - Laziness
- **Parallelism**
  - Data-Parallelism
  - Task-Parallelism
  - Asynchronism

- **Diversity of solutions**
- **Many possible implementations**
- **Adapted for different needs**

Many execution strategies

- **Reactivity**
  - Incrementality
  - Laziness
- **Parallelism**
  - Data-Parallelism
  - Task-Parallelism
  - Asynchronism

- **Diversity of solutions**
- **Many possible implementations**
- **Adapted for different needs**

- **Need of knowledge (expertise)**
- **Need of configuration**
- **<u>Not lowcode friendly</u>**

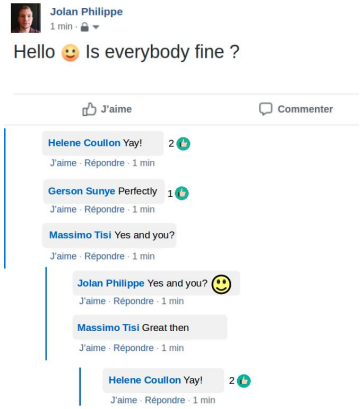
Multi-strategy for parallelism

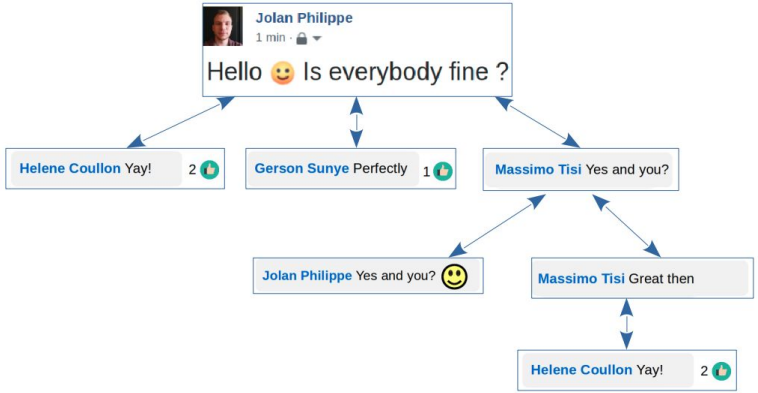
Implementation of strategies using Apache Spark

Support for distributed data structures:

- High-order functions
- MapReduce
- GraphX (+ Pregel)

Model as graphs: A node per class, an edge per relationship.





Direct implementation

1. Get all the comments from a post (recursive DFS)
2. Count the likes (from previous comments)
3. Calculate the score

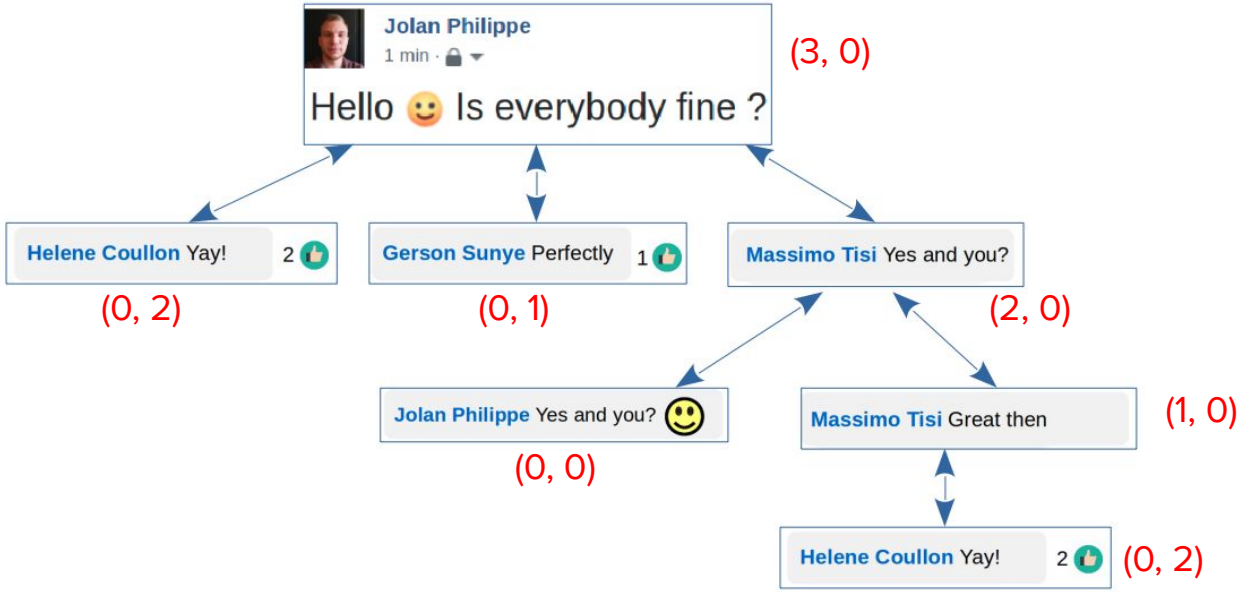
Direct implementation

1. Get all the comments from a post (recursive DFS)
2. Count the likes (from previous comments)
3. Calculate the score

Pros	Cons
Easy to write Easy to read	We do not expect good performances

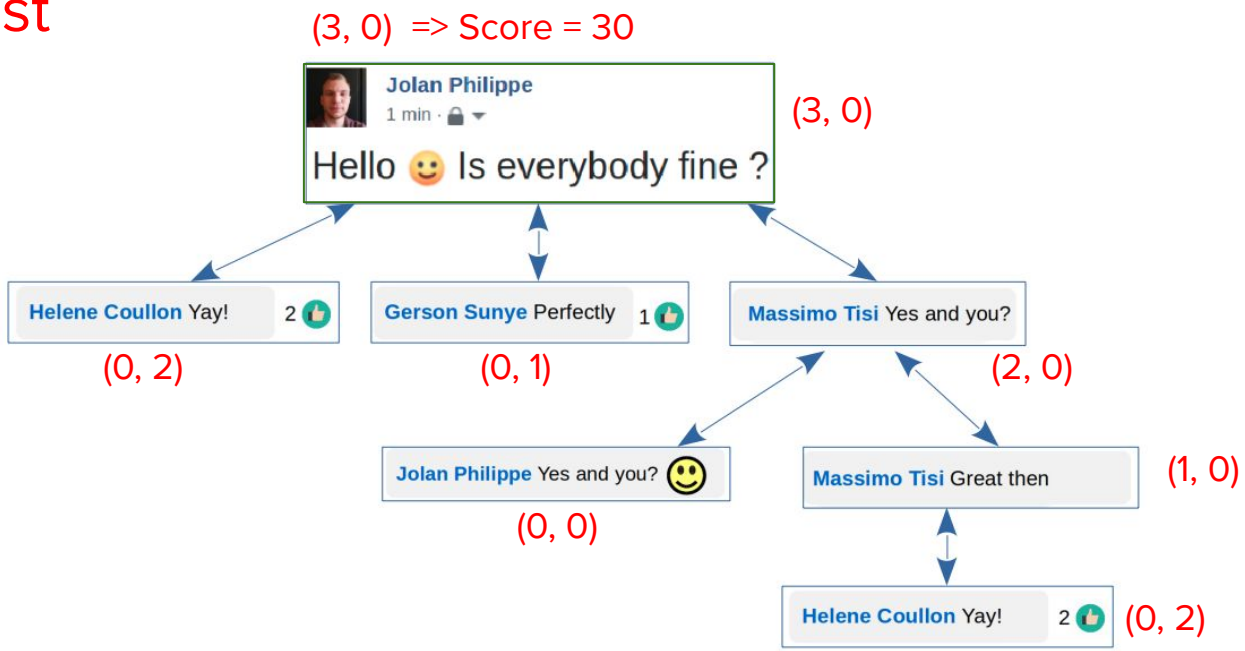
MapReduce implementation

1. Parallel Map + Reduce: a score by submission
2. Accumulation of sub-scores (recursive DFS) for a given post



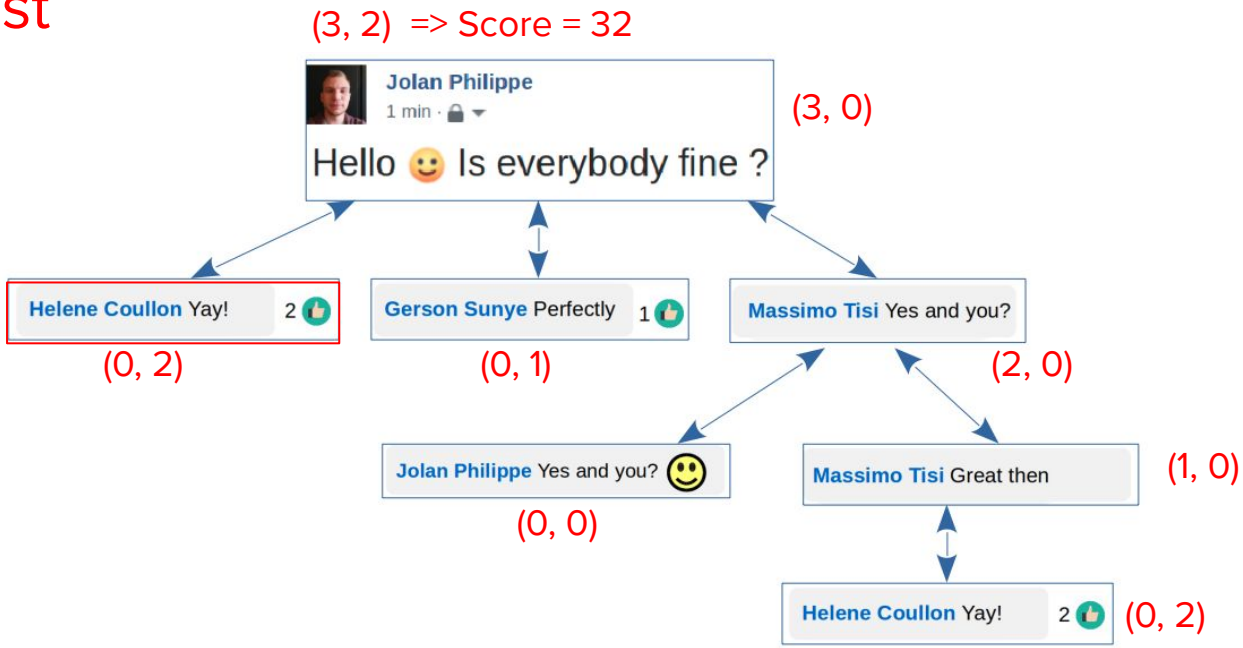
MapReduce implementation

1. Parallel Map + Reduce: a score by submission
2. Accumulation of sub-scores (recursive DFS) for a given post



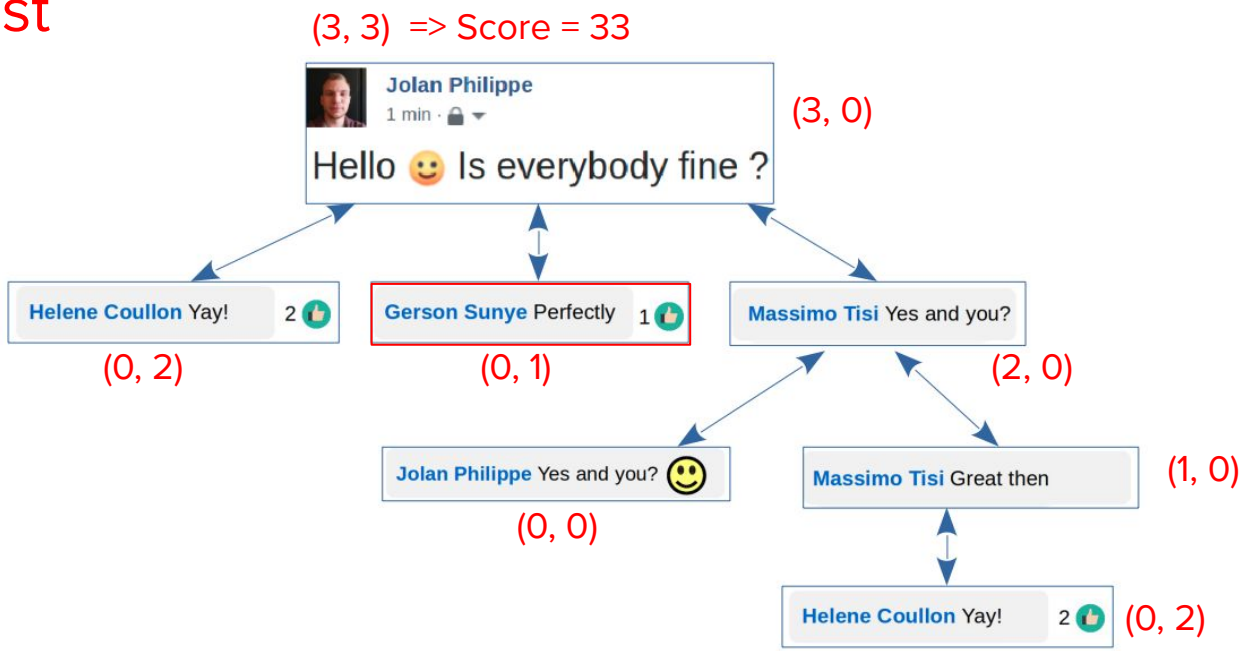
MapReduce implementation

1. Parallel Map + Reduce: a score by submission
2. Accumulation of sub-scores (recursive DFS) for a given post



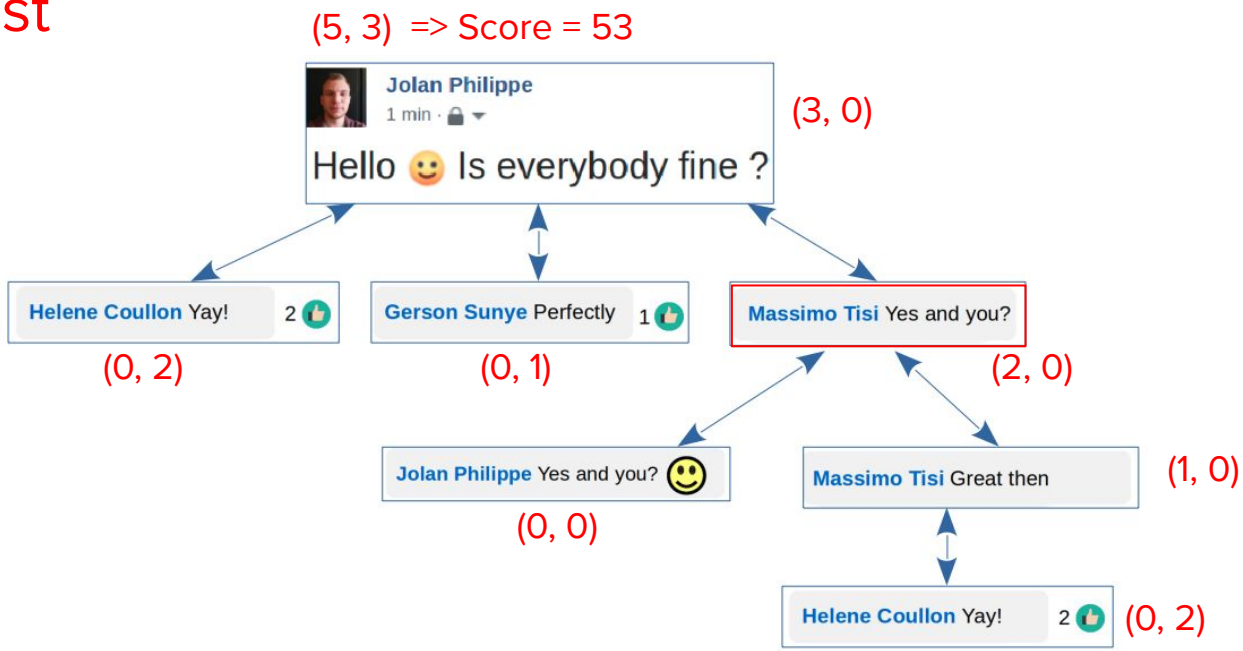
MapReduce implementation

1. Parallel Map + Reduce: a score by submission
2. Accumulation of sub-scores (recursive DFS) for a given post



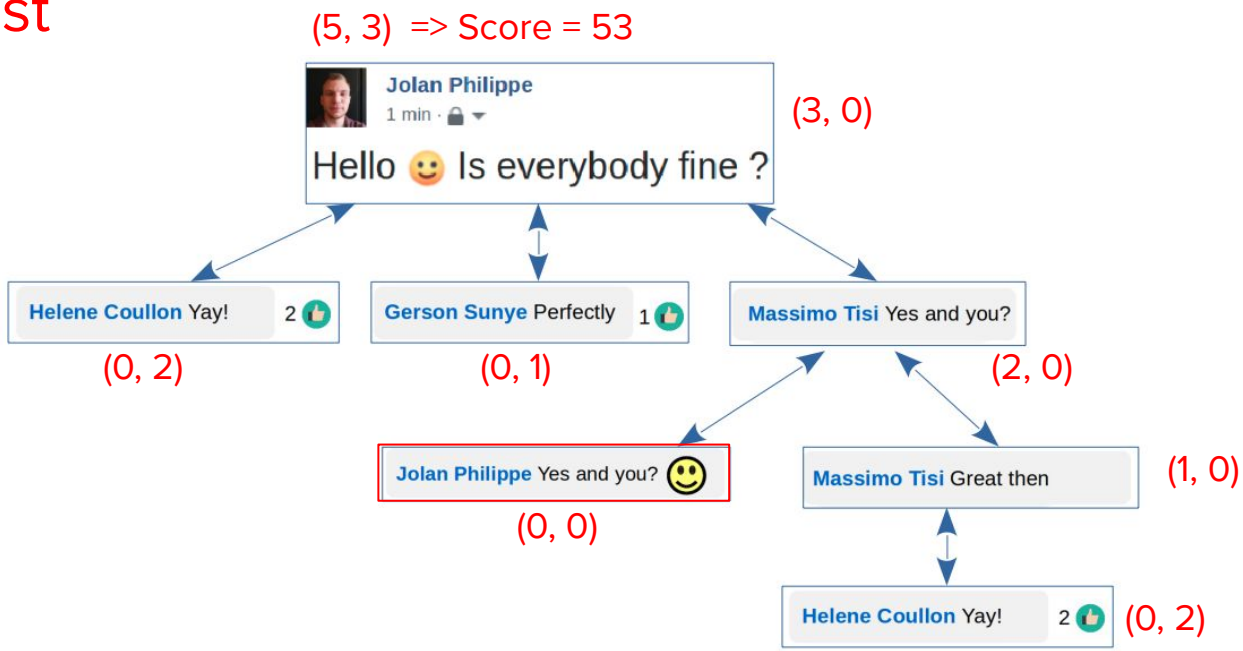
MapReduce implementation

1. Parallel Map + Reduce: a score by submission
2. Accumulation of sub-scores (recursive DFS) for a given post



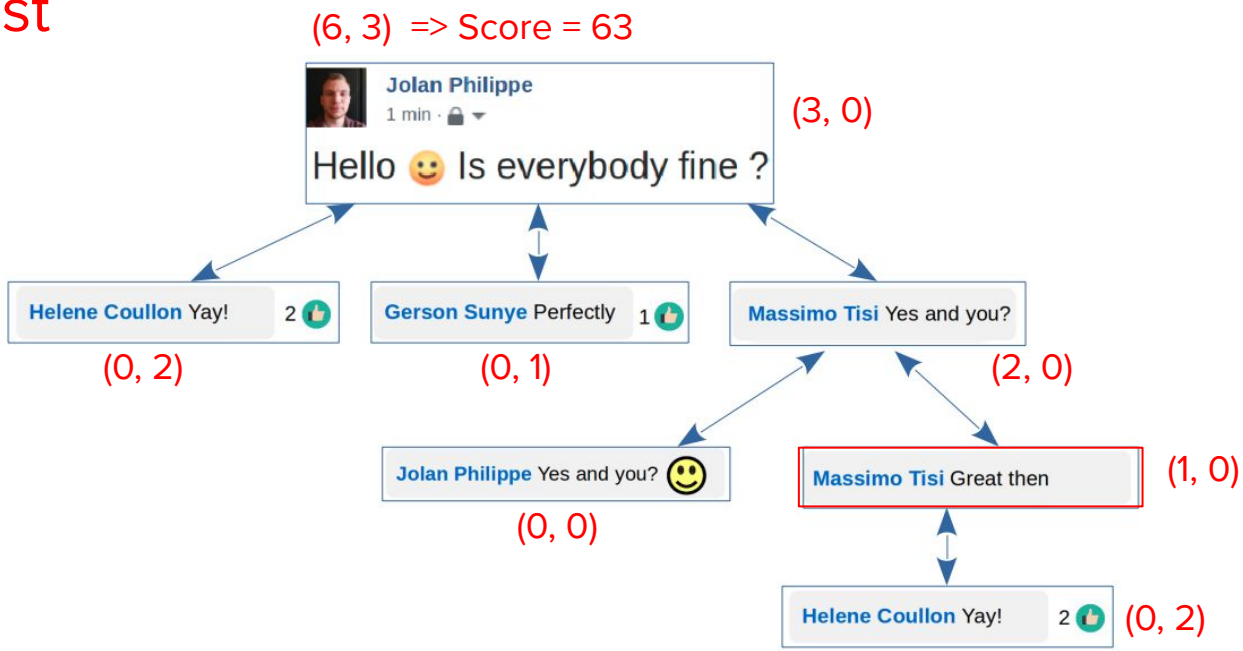
MapReduce implementation

1. Parallel Map + Reduce: a score by submission
2. Accumulation of sub-scores (recursive DFS) for a given post



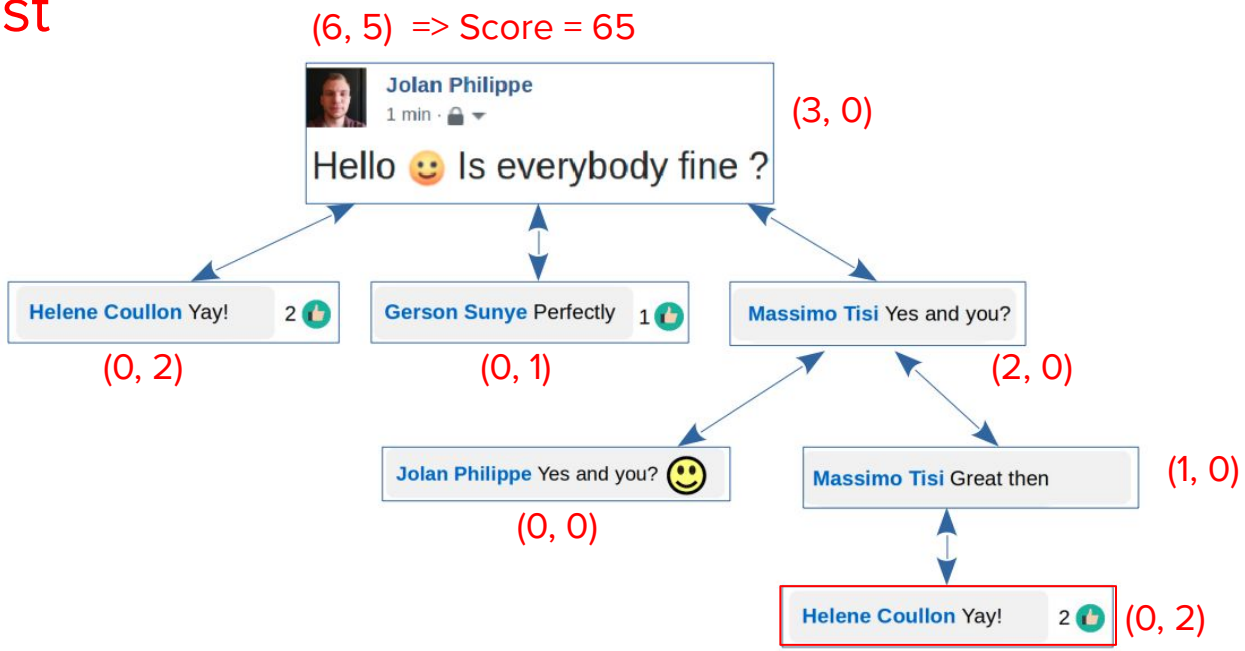
MapReduce implementation

1. Parallel Map + Reduce: a score by submission
2. Accumulation of sub-scores (recursive DFS) for a given post



MapReduce implementation

1. Parallel Map + Reduce: a score by submission
2. Accumulation of sub-scores (recursive DFS) for a given post



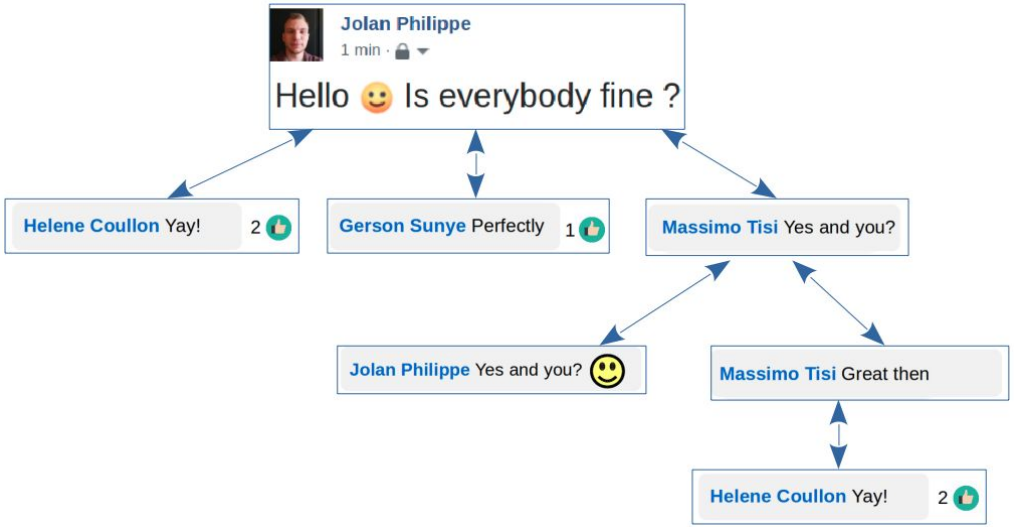
MapReduce implementation

1. Parallel Map + Reduce: a score by submission
2. Accumulation of sub-scores (recursive DFS) for a given post

Pros	Cons
Highly parallel solution	Not fit for frequently modified model

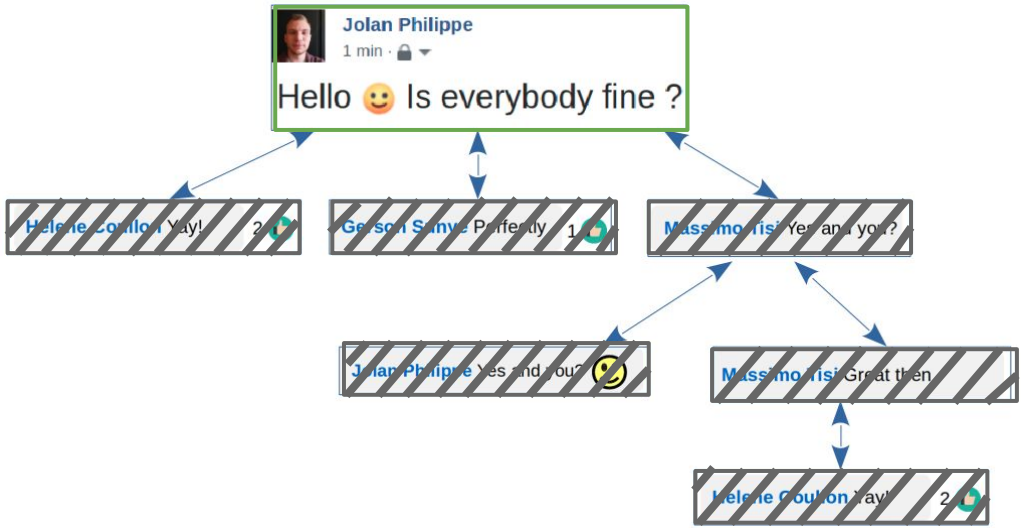
Pregel implementation

1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
   a. message passing
   b. with accumulators
3. Calculate the score from accumulators



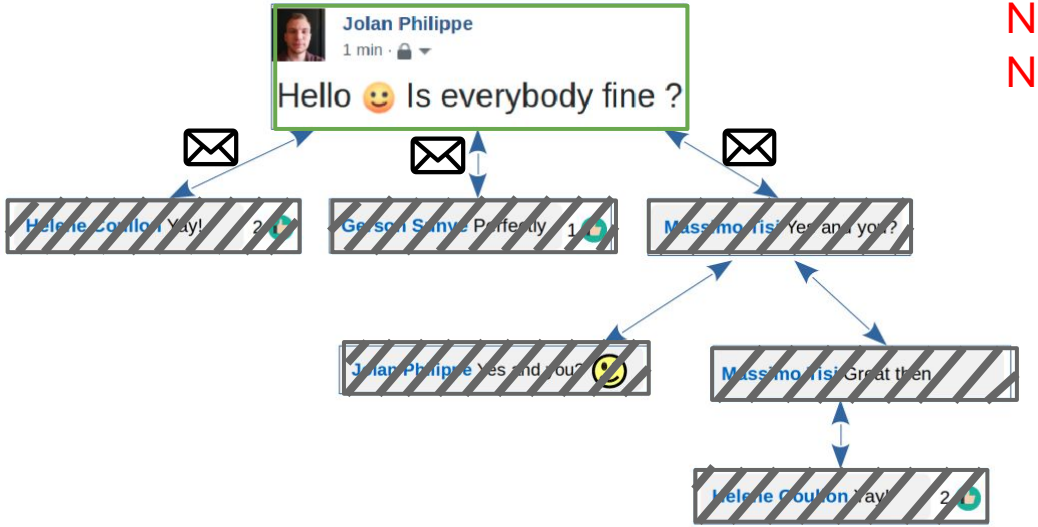
Pregel implementation

1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
   a. message passing
   b. with accumulators
3. Calculate the score from accumulators



Pregel implementation

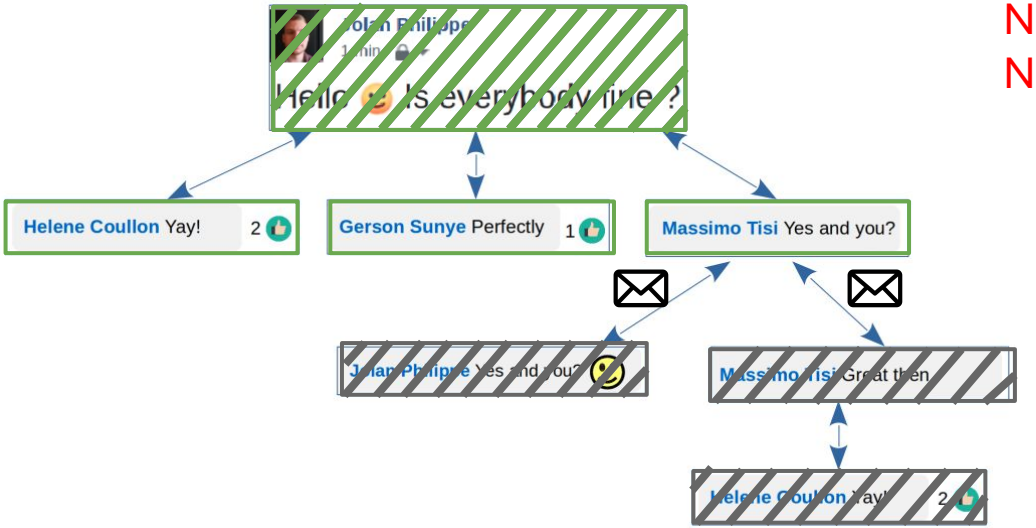
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
   a. message passing
   b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 0
Nb_likes = 0

Pregel implementation

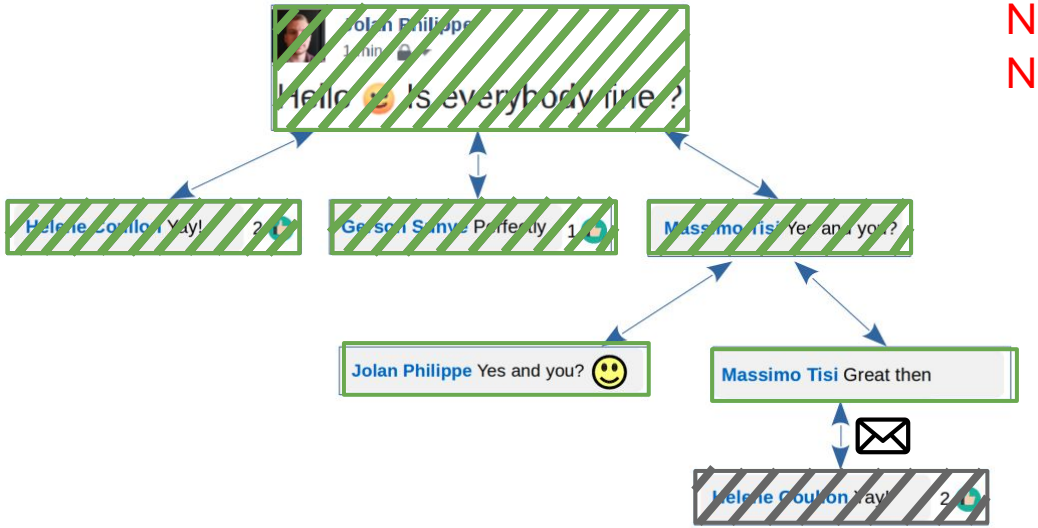
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
   a. message passing
   b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 3
Nb_likes = 3

Pregel implementation

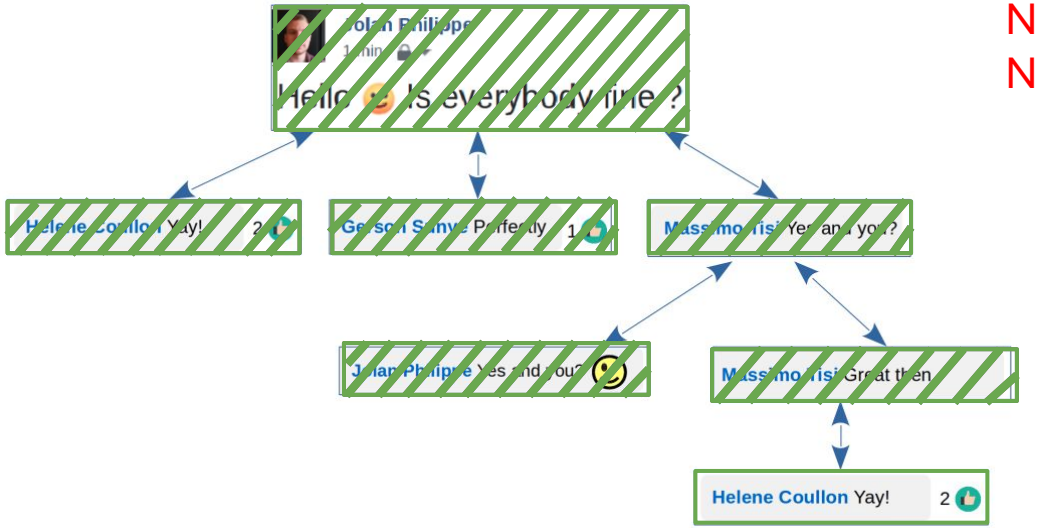
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
   a. message passing
   b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 5
Nb_likes = 3

Pregel implementation

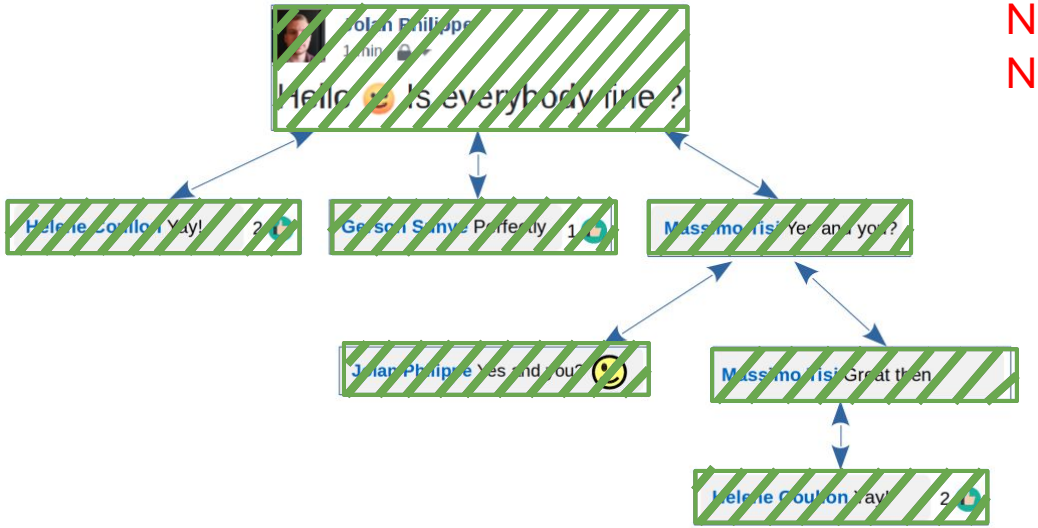
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
   a. message passing
   b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 6
Nb_likes = 5

Pregel implementation

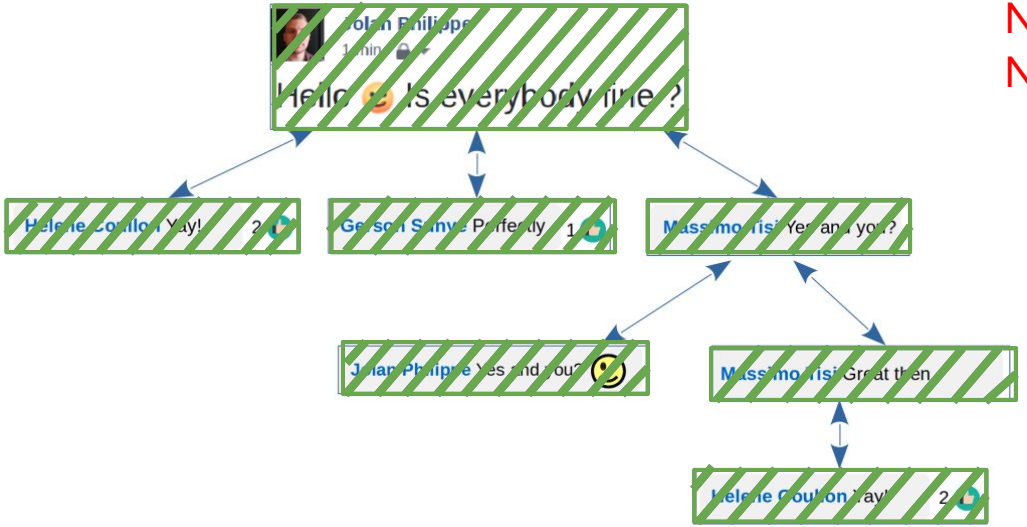
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
   a. message passing
   b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 6
Nb_likes = 5

Pregel implementation

1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
   a. message passing
   b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 6
Nb_likes = 5
= > Score = 65

Pregel implementation

1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
   a. message passing
   b. with accumulators
3. Calculate the score from accumulators

Pros	Cons
Easy to write	Hard to read
Strong engine (perf)	Limited parallelism

Multi-Strategy implementation

- **Direct implementation + Pregel**
  1. Parallel Map + Reduce: a score by comment
  2. Accumulation of sub-scores (DFS to Pregel) for a given post

- **MapReduce implementation + Pregel**
  1. Get all the comments from a post (DFS to Pregel)
  2. Count the likes (from previous comments)
  3. Calculate the score

Experiments

Execution of the score functions on all posts:

- 5 Strategies
- 30 executions each
- 8 datasets ((1274 nodes, 2533 edges) to (115121 nodes, 286502 edges))

Single machine configuration:

- Java 1.8 with Scala 2.13.2 (Spark 3.0.1)
- Intel(R) Core(TM) i7-8650U CPU
  - 8 cores
  - 1.90GHz
- Memory of 32 GB

Results

	Dataset				Speed-up (compared to Naive Sequential)					
#	# users	# posts	# comments	# likes	Naive Sequential	Naive Parallel	Pregel	MapReduce	OCL + Pregel	MapReduce + Pregel
1	80	554	640	6	**1x**	0.40x	10.30x	5.82x	9.40x	4.63x
2	889	1064	118	24	**1x**	0.39x	0.36x	0.46x	0.44x	0.46x
3	1845	2315	190	66	**1x**	0.51x	0.68x	0.85x	0.66x	0.71x
4	2270	5056	204	129	**1x**	0.27x	0.35x	2.34x	0.15x	2.96x
5	5518	9220	394	572	**1x**	4.25x	5.21x	4.17x	4.68x	4.03x
6	10929	18872	595	1598	**1x**	4.68x	2.83x	2.39x	1.97x	3.91x
7	18083	39212	781	4770	**1x**	4.07x	4.12x	4.58x	5.17x	3.27x
8	37228	76735	1158	13374	**1x**	7.28x	9.52x	7.61x	9.66x	9.22x

Results

	Dataset				Speed-up (compared to Naive Sequential)					
#	# users	# posts	# comments	# likes	Naive Sequential	Naive Parallel	Pregel	MapReduce	OCL + Pregel	MapReduce + Pregel
1	80	554	640	6	**1x**	0.40x	10.30x	5.82x	9.40x	4.63x
2	889	1064	118	24	**1x**	0.39x	0.36x	0.46x	0.44x	0.46x
3	1845	2315	190	66	**1x**	0.51x	0.68x	0.85x	0.66x	0.71x
4	2270	5056	204	129	**1x**	0.27x	0.35x	2.34x	0.15x	2.96x
5	5518	9220	394	572	**1x**	4.25x	5.21x	4.17x	4.68x	4.03x
6	10929	18872	595	1598	**1x**	4.68x	2.83x	2.39x	1.97x	3.91x
7	18083	39212	781	4770	**1x**	4.07x	4.12x	4.58x	5.17x	3.27x
8	37228	76735	1158	13374	**1x**	7.28x	9.52x	7.61x	9.66x	9.22x

Results

	Dataset				Speed-up (compared to Naive Sequential)					
#	# users	# posts	# comments	# likes	Naive Sequential	Naive Parallel	Pregel	MapReduce	OCL + Pregel	MapReduce + Pregel
1	80	554	640	6	**1x**	0.40x	10.30x	5.82x	9.40x	4.63x
2	889	1064	118	24	**1x**	0.39x	0.36x	0.46x	0.44x	0.46x
3	1845	2315	190	66	**1x**	0.51x	0.68x	0.85x	0.66x	0.71x
4	2270	5056	204	129	**1x**	0.27x	0.35x	2.34x	0.15x	2.96x
5	5518	9220	394	572	**1x**	4.25x	5.21x	4.17x	4.68x	4.03x
6	10929	18872	595	1598	**1x**	4.68x	2.83x	2.39x	1.97x	3.91x
7	18083	39212	781	4770	**1x**	4.07x	4.12x	4.58x	5.17x	3.27x
8	37228	76735	1158	13374	**1x**	7.28x	9.52x	7.61x	9.66x	9.22x

Solution

- Several strategies for model management: Example with <u>parallelism</u>

  Not all adapted for every situation
- Need of additional metadata
  - Size and topology of model
  - Kind of operation and their frequency
  - (Available architecture)
- Adaptive engine based on a multi-strategy approach

Conclusion

- Many execution strategies
- As a concrete example: several parallel strategies (TTC18)
- There is no solution adapted in every situation
- Introduction of a multi-strategy engine

Conclusion

- Many execution strategies
- As a concrete example: several parallel strategies (TTC18)
- There is no solution adapted in every situation
- Introduction of a multi-strategy engine

Perspective and future work

- Implement and experiment reactive (incremental and lazy) aspects
- Conduct additional experiments:
  - Larger dataset
  - Distributed architecture
  - Specific topologies

Conclusion

- Many execution strategies
- As a concrete example: several parallel strategies (TTC18)
- There is no solution adapted in every situation
- Introduction of a multi-strategy engine

Perspective and future work

- Implement and experiment reactive (incremental and lazy) aspects
- Conduct additional experiments:
  - Larger dataset
  - Distributed architecture
  - Specific topologies

Questions ?