

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

Jolan Philippe¹², H el ene Coullon¹²³, Massimo Tisi¹², and Gerson Suny e²⁴
IMT Atlantique¹, LS2N², Inria³, Universit e de Nantes⁴

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



Jolan Philippe

1 min · 🔒

Hello 😊 Is everybody fine ?

👍 J'aime

💬 Commenter

Helene Coullon Yay! 2 👍

J'aime · Répondre · 1 min

Gerson Sunye Perfectly 1 👍

J'aime · Répondre · 1 min

Massimo Tisi Yes and you?

J'aime · Répondre · 1 min

Jolan Philippe Yes and you? 😊

J'aime · Répondre · 1 min

Massimo Tisi Great then

J'aime · Répondre · 1 min

Helene Coullon Yay! 2 👍

J'aime · Répondre · 1 min

- Query:
What is the most debated post in a social network?

Transformation Tool Contest 2018



Jolan Philippe

1 min · 🔒

Hello 😊 Is everybody fine ?

👍 J'aime

💬 Commenter

Helene Coullon Yay!

2 👍

J'aime · Répondre · 1 min

Gerson Sunye Perfectly

1 👍

J'aime · Répondre · 1 min

Massimo Tisi Yes and you?

J'aime · Répondre · 1 min

Jolan Philippe Yes and you? 😊

J'aime · Répondre · 1 min

Massimo Tisi Great then

J'aime · Répondre · 1 min

Helene Coullon Yay!

2 👍

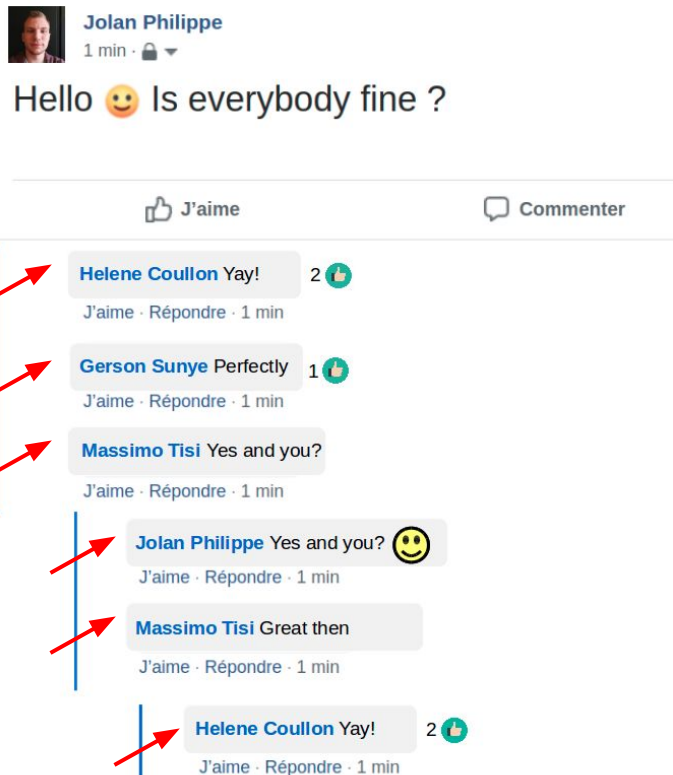
J'aime · Répondre · 1 min

- Query:

What is the most debated post in a social network?

$$\text{Score}(\text{post}) = 10 * \#(\text{post.comments}) + \#(\text{post.comments.likes})$$

Transformation Tool Contest 2018

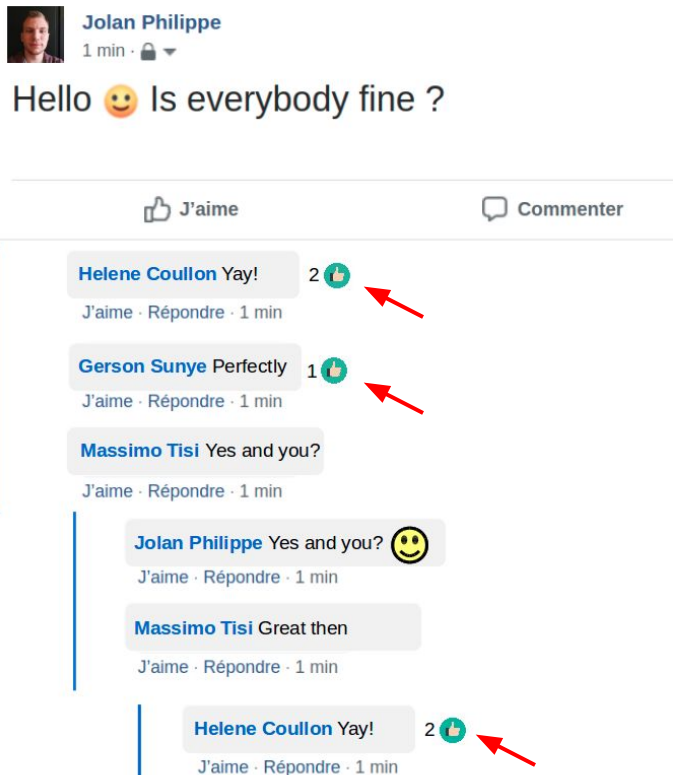


The screenshot shows a social media post by Johan Philippe. The post text is "Hello 😊 Is everybody fine ?". Below the post are interaction buttons for "J'aime" and "Commenter". There are six replies, each with a red arrow pointing to it from the left. The replies are:

- Helene Coullon: "Yay!" (2 likes)
- Gerson Sunye: "Perfectly" (1 like)
- Massimo Tisi: "Yes and you?"
- Johan Philippe: "Yes and you? 😊" (1 like)
- Massimo Tisi: "Great then"
- Helene Coullon: "Yay!" (2 likes)

- 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



The screenshot shows a social media post by Johan Philippe. The post text is "Hello 😊 Is everybody fine ?". Below the post are two buttons: "J'aime" (with a thumbs up icon) and "Commenter" (with a speech bubble icon). There are five comments visible, each with a "J'aime" button and a "Répondre" button. The comments are: Helene Coullon "Yay!" (2 likes), Gerson Sunye "Perfectly" (1 like), Massimo Tisi "Yes and you?", Johan Philippe "Yes and you?" (with a smiley face emoji), and Massimo Tisi "Great then". At the bottom, another comment by Helene Coullon "Yay!" (2 likes) is partially visible. Red arrows point to the like icons on the first, second, and bottom-most comments.

- Query:

What is the most debated post in a social network?

$$\text{Score}(\text{post}) = 10 * \#(\text{post.comments}) + \#(\text{post.comments.likes})$$

- $\#(\text{post.comments}) = 6$
- $\#(\text{post.comments.likes}) = 5$

Transformation Tool Contest 2018



Johan Philippe

1 min · 🔒

Hello 😊 Is everybody fine ?

👍 J'aime

💬 Commenter

Helene Coullon Yay! 2 👍

J'aime · Répondre · 1 min

Gerson Sunye Perfectly 1 👍

J'aime · Répondre · 1 min

Massimo Tisi Yes and you?

J'aime · Répondre · 1 min

Johan Philippe Yes and you? 😊

J'aime · Répondre · 1 min

Massimo Tisi Great then

J'aime · Répondre · 1 min

Helene Coullon Yay! 2 👍

J'aime · Répondre · 1 min

- Query:

What is the most debated post in a social network?

$$\text{Score}(\text{post}) = 10 * \#(\text{post.comments}) + \#(\text{post.comments.likes})$$

- $\#(\text{post.comments}) = 6$
- $\#(\text{post.comments.likes}) = 5$
- $\text{Score}(\text{post}) = 65$

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**

- **Not lowcode friendly**

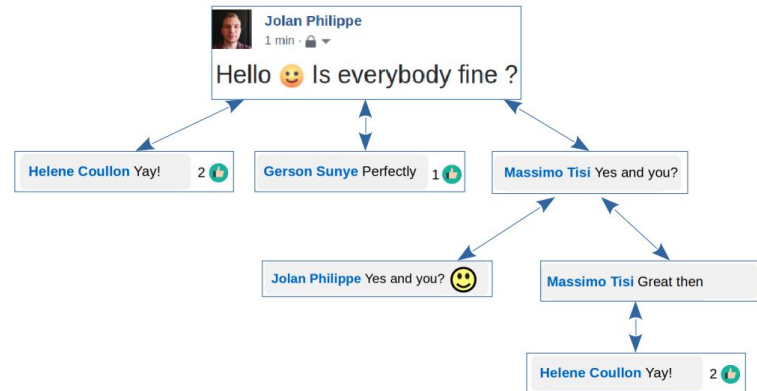
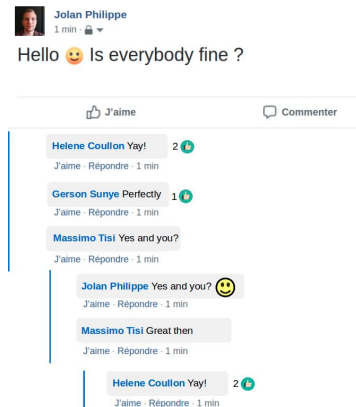
Multi-strategy for parallelism

Implementation of strategies using 

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

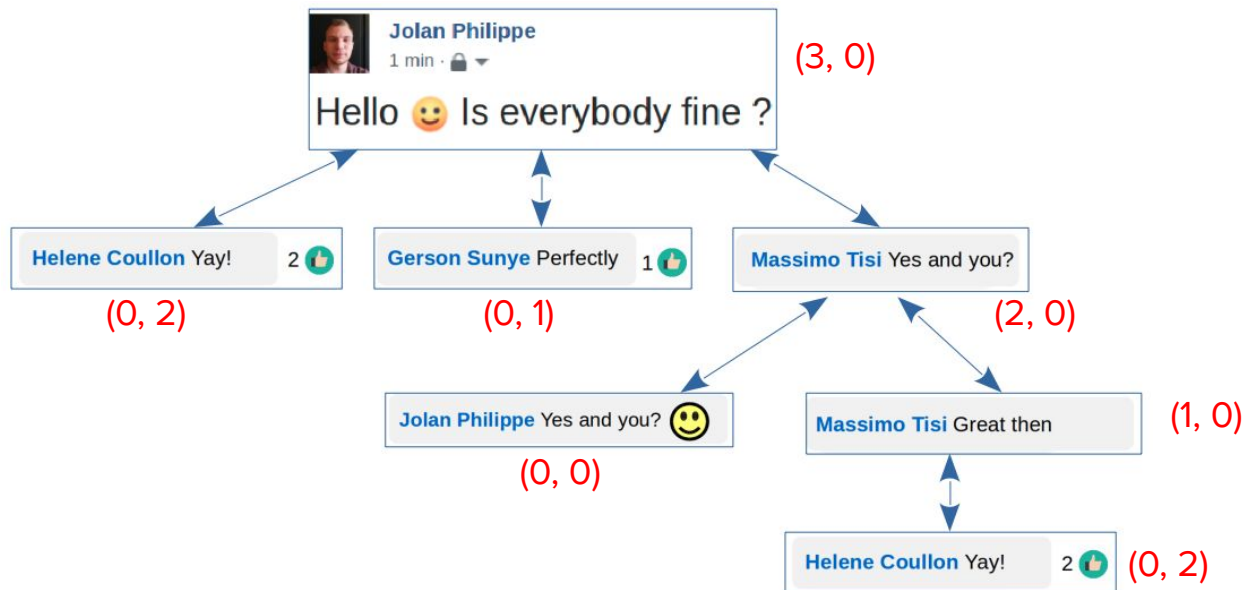
Easy to write
Easy to read

Cons

We do not expect
good
performances

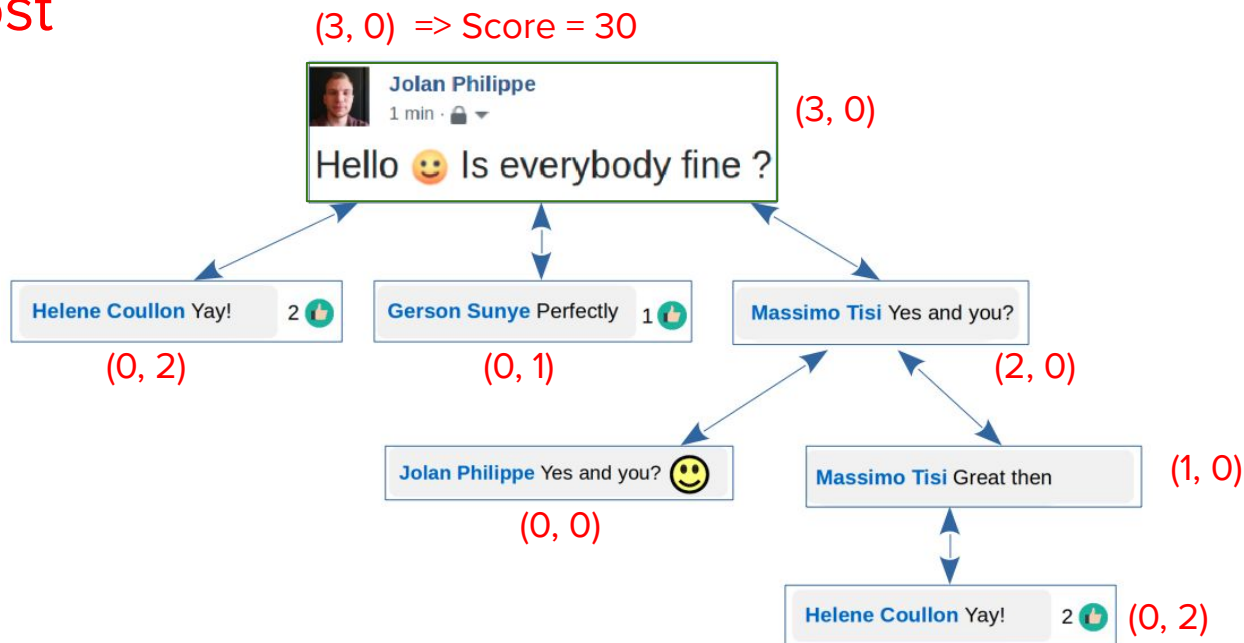
MapReduce implementation

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



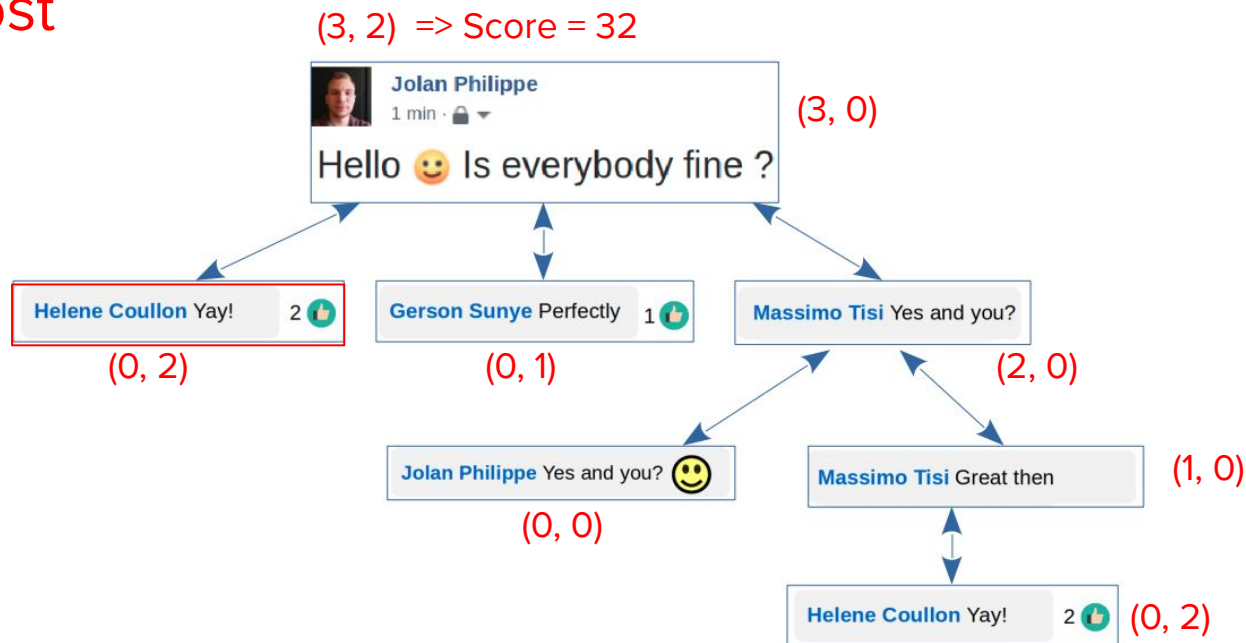
MapReduce implementation

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



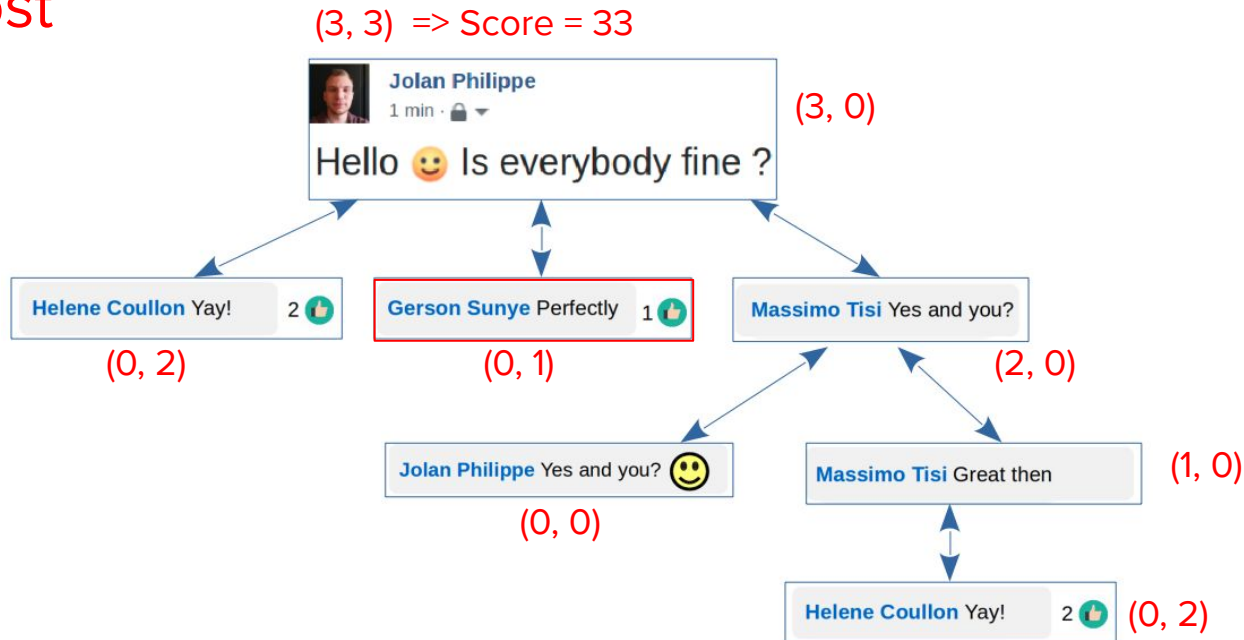
MapReduce implementation

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



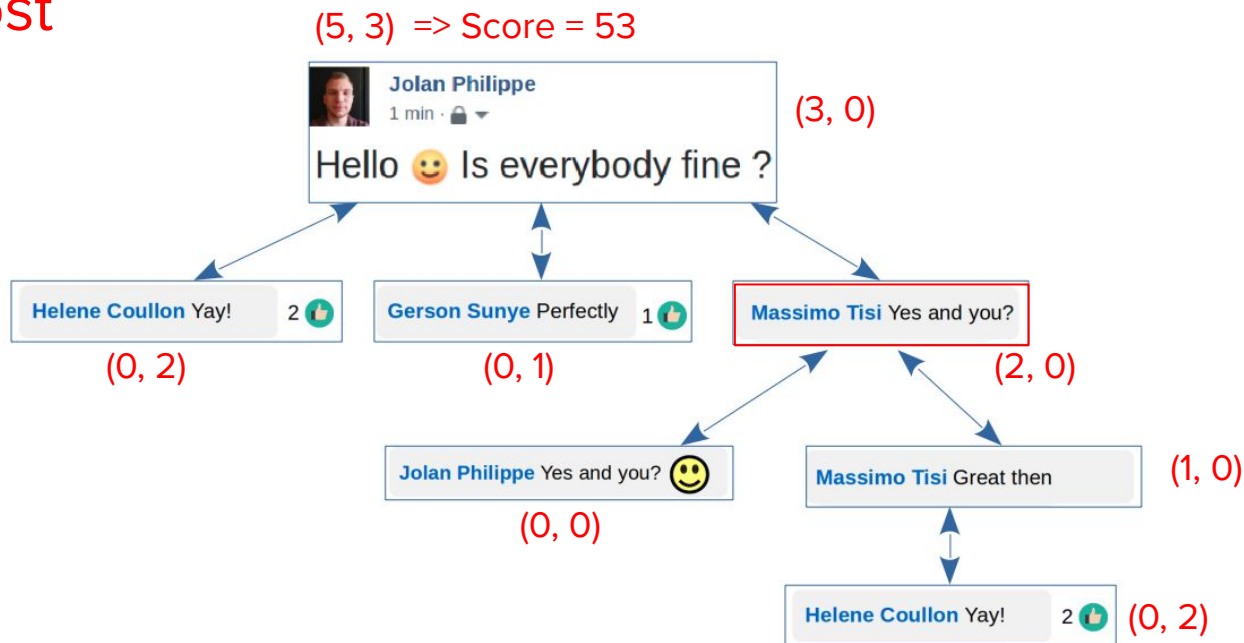
MapReduce implementation

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



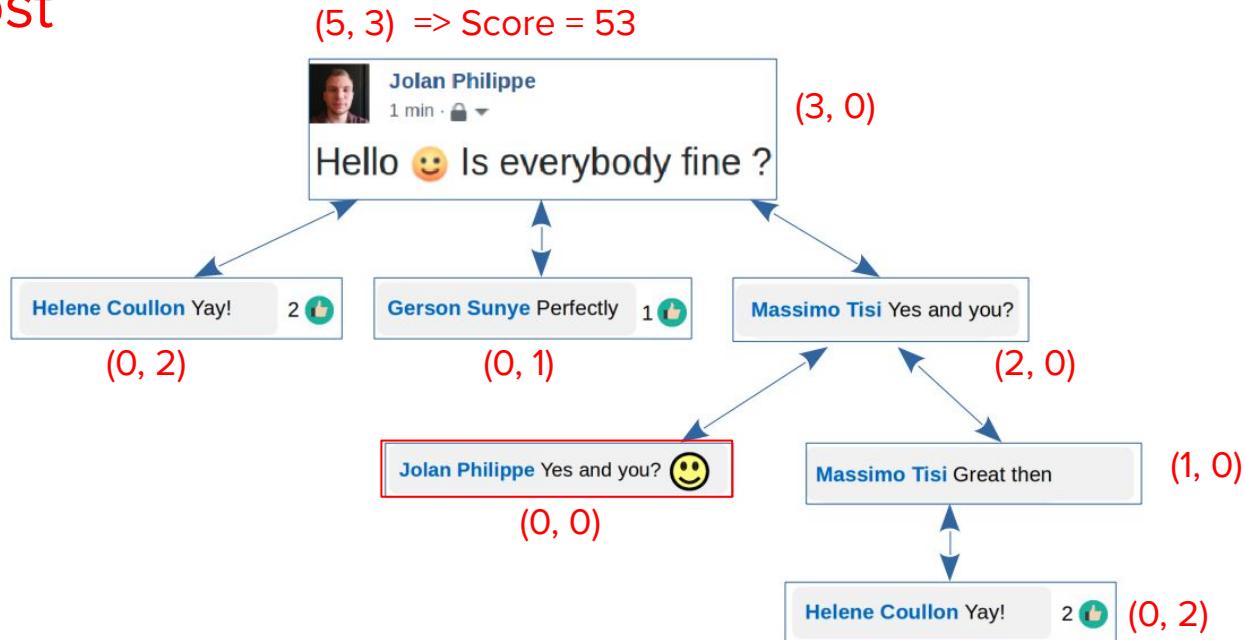
MapReduce implementation

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



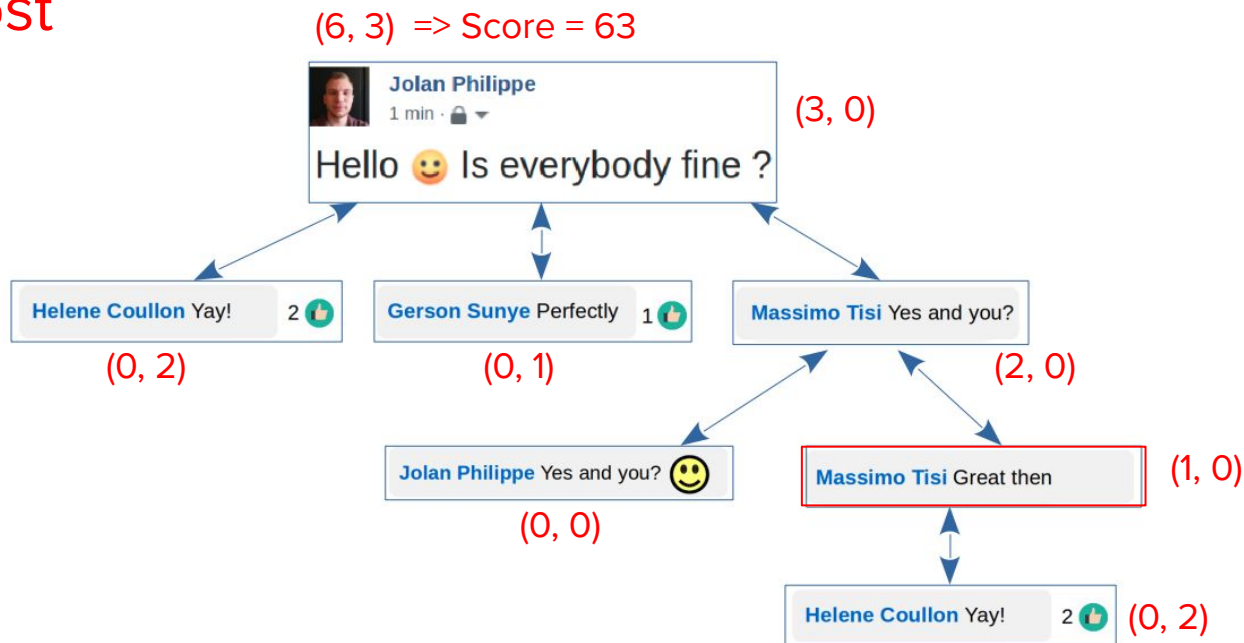
MapReduce implementation

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



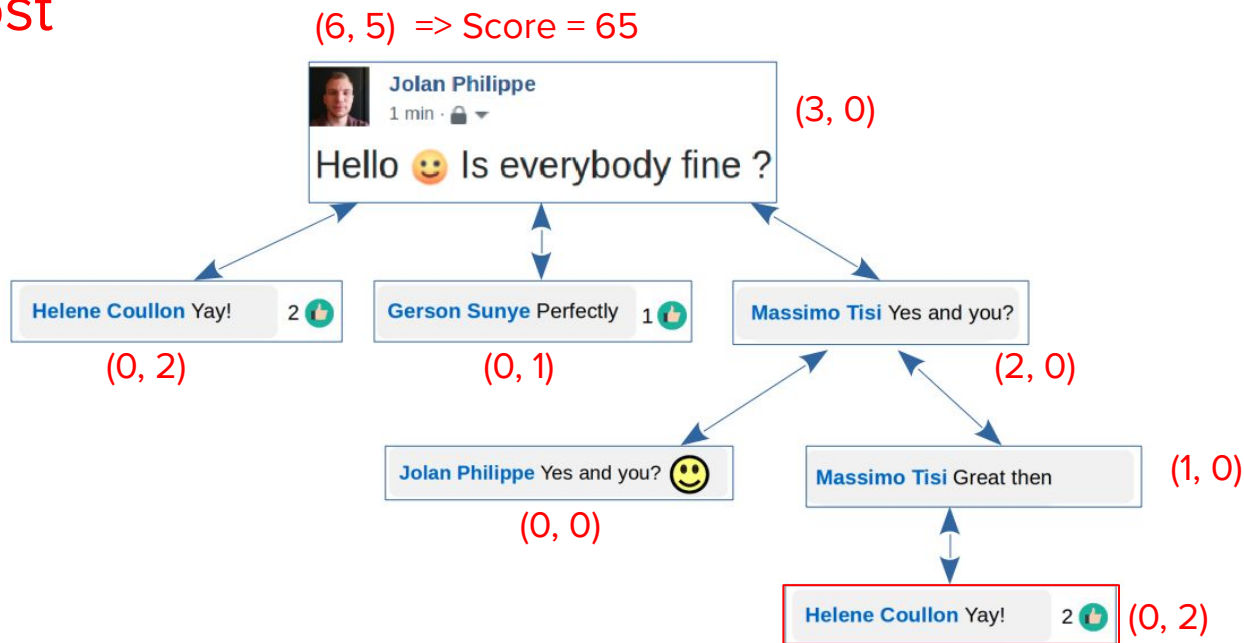
MapReduce implementation

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

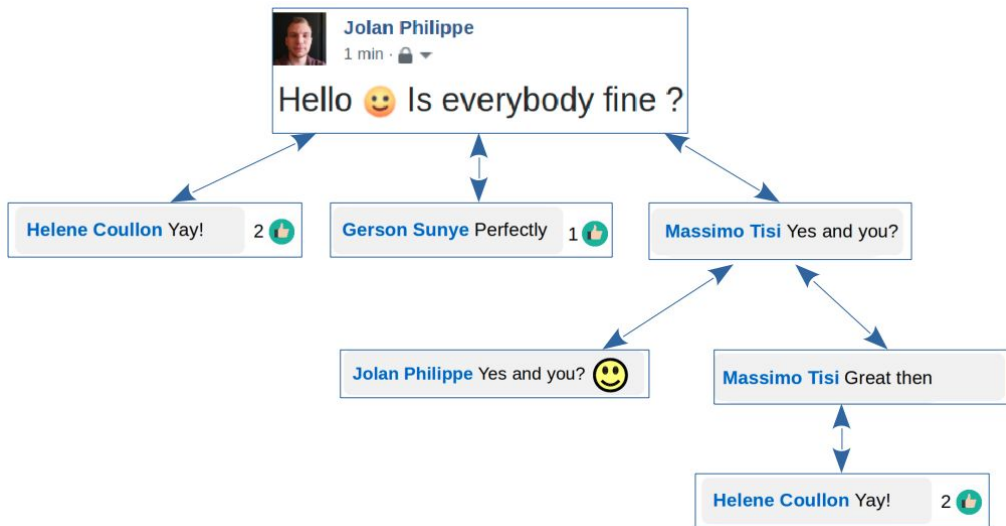
Highly parallel
solution

Cons

Not fit for frequently
modified model

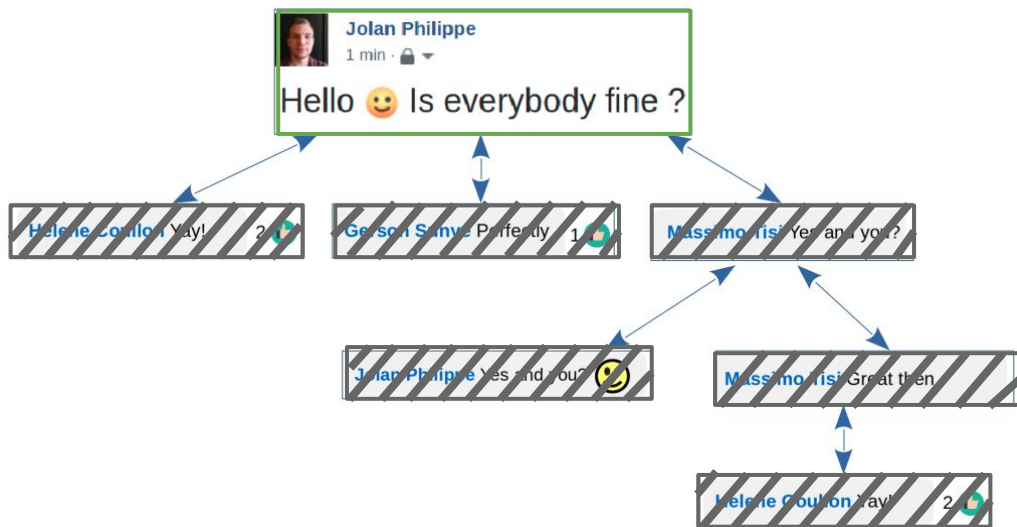
Pregel implementation

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



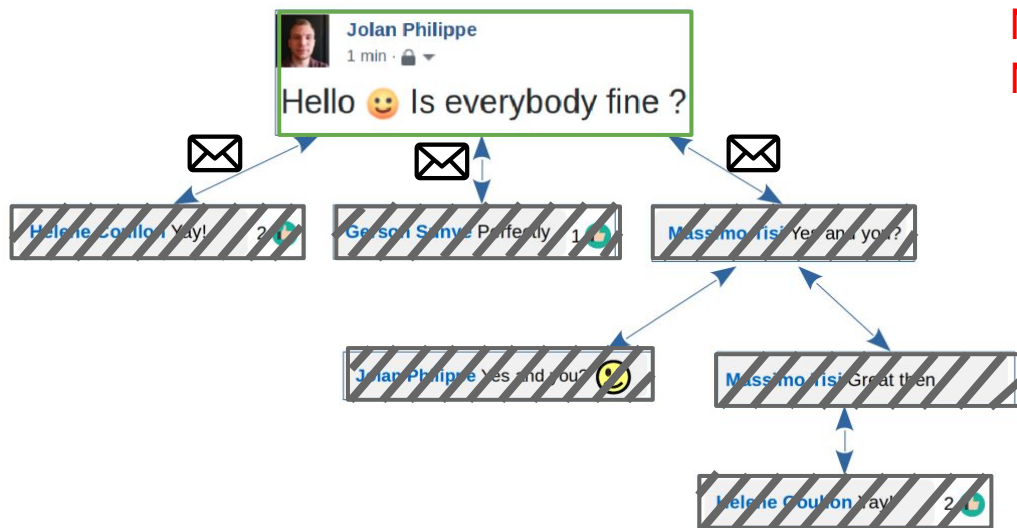
Pregel implementation

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

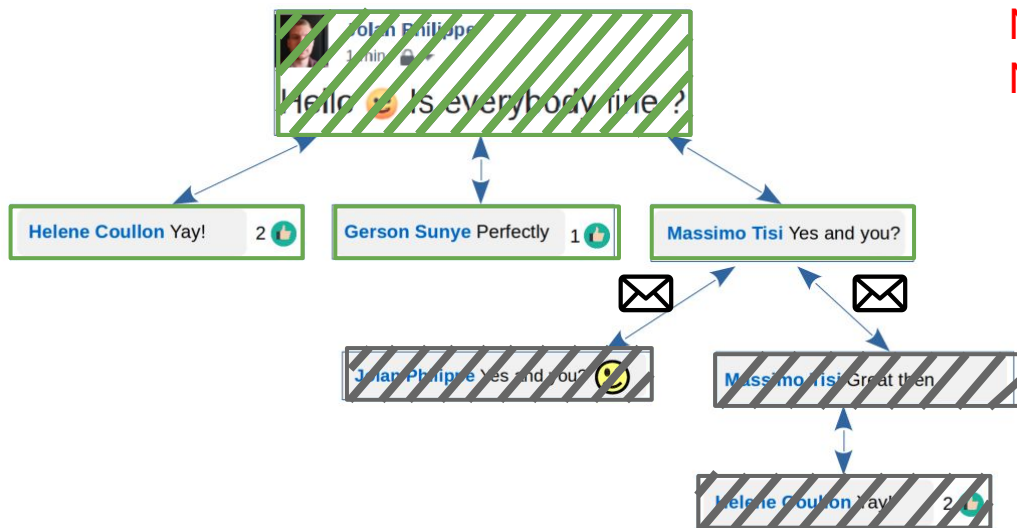
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
 - a. message passing
 - b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 0
Nb_likes = 0

Pregel implementation

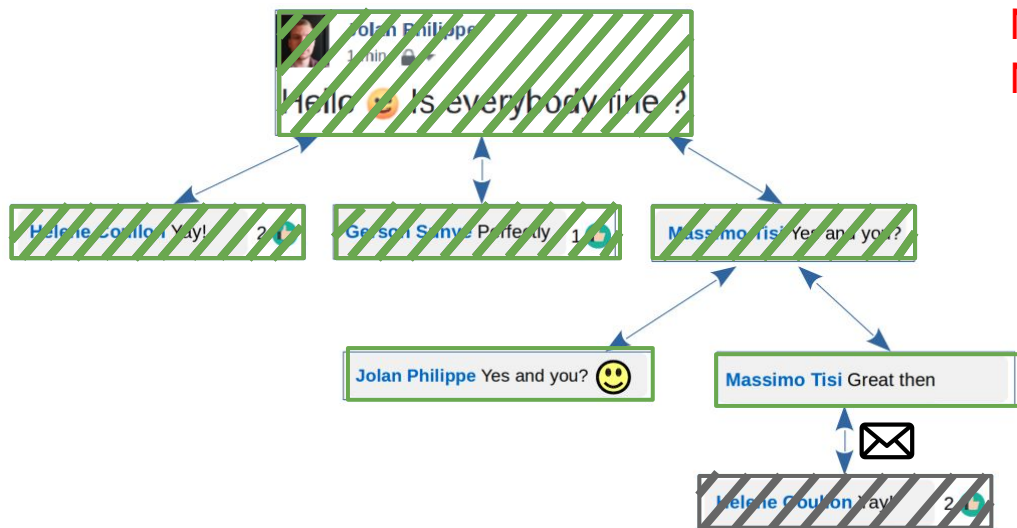
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
 - a. message passing
 - b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 3
Nb_likes = 3

Pregel implementation

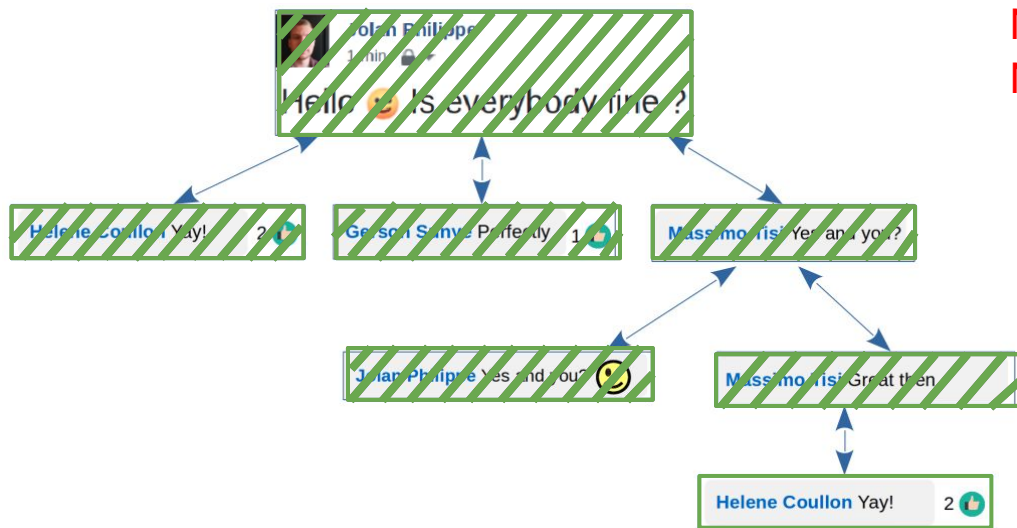
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
 - a. message passing
 - b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 5
Nb_likes = 3

Pregel implementation

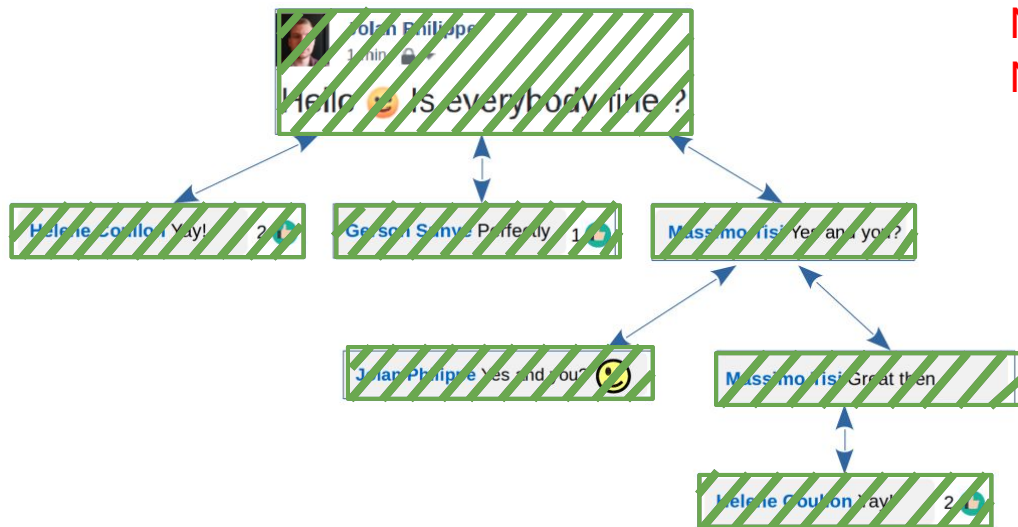
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
 - a. message passing
 - b. with accumulators
3. Calculate the score from accumulators



Nb_comments = 6
Nb_likes = 5

Pregel implementation

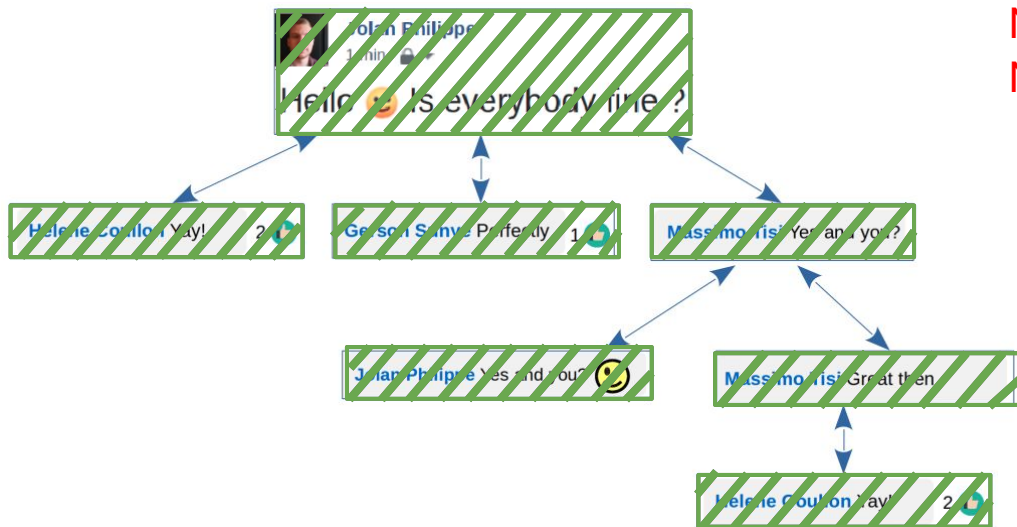
1. Initial step: parallel Map
2. Reachability by propagation (Pregel) from a post
 - a. message passing
 - b. with accumulators
3. Calculate the score from accumulators



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

Easy to write

Strong engine (perf)

Cons

Hard to read

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 parallelism
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 ?