# Distributed Stable Marriage with Incomplete List and Ties using Spark

Yilong Geng  Mingyu Gao

Stanford University

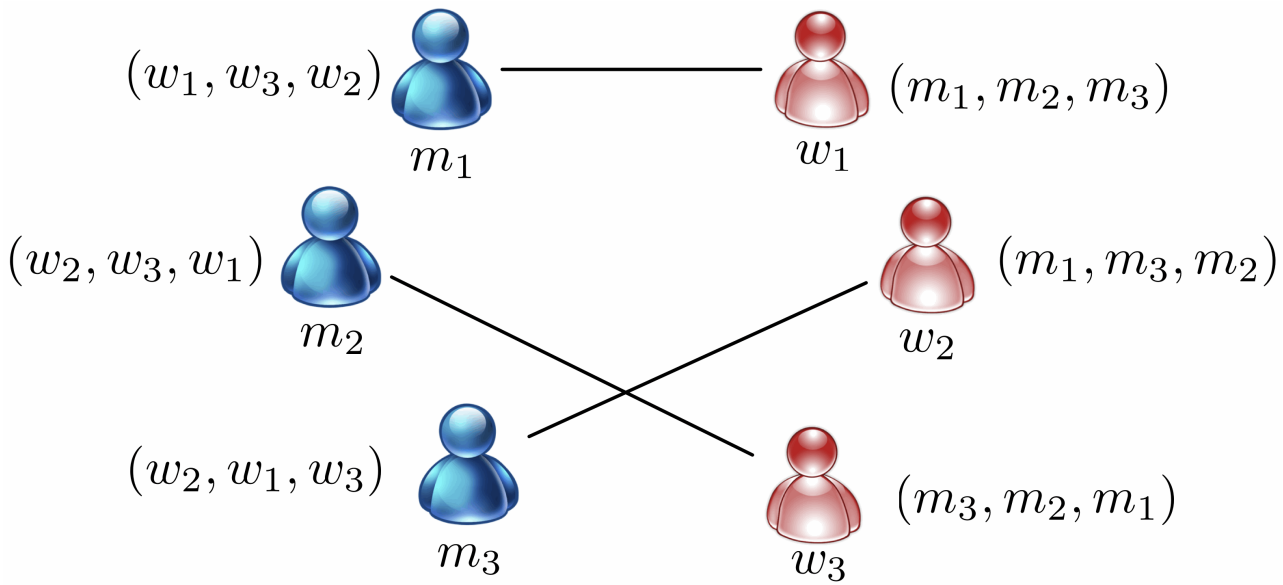
{gengyl08,mgao12}@stanford.edu

June 1, 2015

# Overview

# Marriage Problem





# Stable Marriage with Incomplete list and Ties (SMTI)



## Blocking Pair

A pair $mw$ is blocking $M$, if $mw \in E \backslash M$ (they are an acceptable pair and they are not matched) and

- $w$ is not engaged or strictly prefers $m$ to her fiance, **and**
- $m$ is not engaged or strictly prefers $w$ to his fiancee.

## Stable

A marriage is called stable if there is no blocking pair.

## Incomplete list and Ties

Preference list can be incomplete, and/or have ties

**Our goal**: find a stable marriage with maximum marriage size

# Basic GS Algorithm



```
1: Initialize all m and w to free
2: while there is a free m who has non-empty list do
3:     w = top in the list
4:     if w is free then
5:         mw engage
6:     else
7:         Some m'w already engaged
8:         if w prefers m to m' then
9:             mw engage, m' becomes free
10:            Remove w from m''s list
11:        else
12:            m'w remain engaged, remove w from m's list
13:        end if
14:    end if
15: end while
```

- Single machine complexity: $O(e)$
- Result size $\geq \frac{1}{2}$ of the maximum size

# Modified GS for SMTI



- Zoltan Kiraly, *Linear Time Local Approximation Algorithm for Maximum Stable Marriage*.
- Key modification
  - Men will go through his list twice
  - Break tie with the current status of the men and women
- Single machine complexity: $O(e)$
- Result size $\geq \frac{2}{3}$ of the maximum size

# Distributed Modified GS



- Preference list length is usually much shorter than $n$.
- Proposers (men) and acceptors (women) are represented in RDD, including their preference lists.
- In each iteration, all proposers will propose simultaneously, and acceptors pick the best proposals.
- Communication complexity: $O(n)$ per iteration.
- Single worker reduce complexity: maximum list length.

# Use Pregel?



- Two groups of people form a (sparse) bipartite graph, with (incomplete) preference lists as edges
- Advantages
  - Pregel handles partitioning better
  - Simple API
- Problems
  - Different attributes for proposers/acceptors
  - Each vertex only sends message along one edge in each iteration
  - Two rounds of message transfer in one iteration

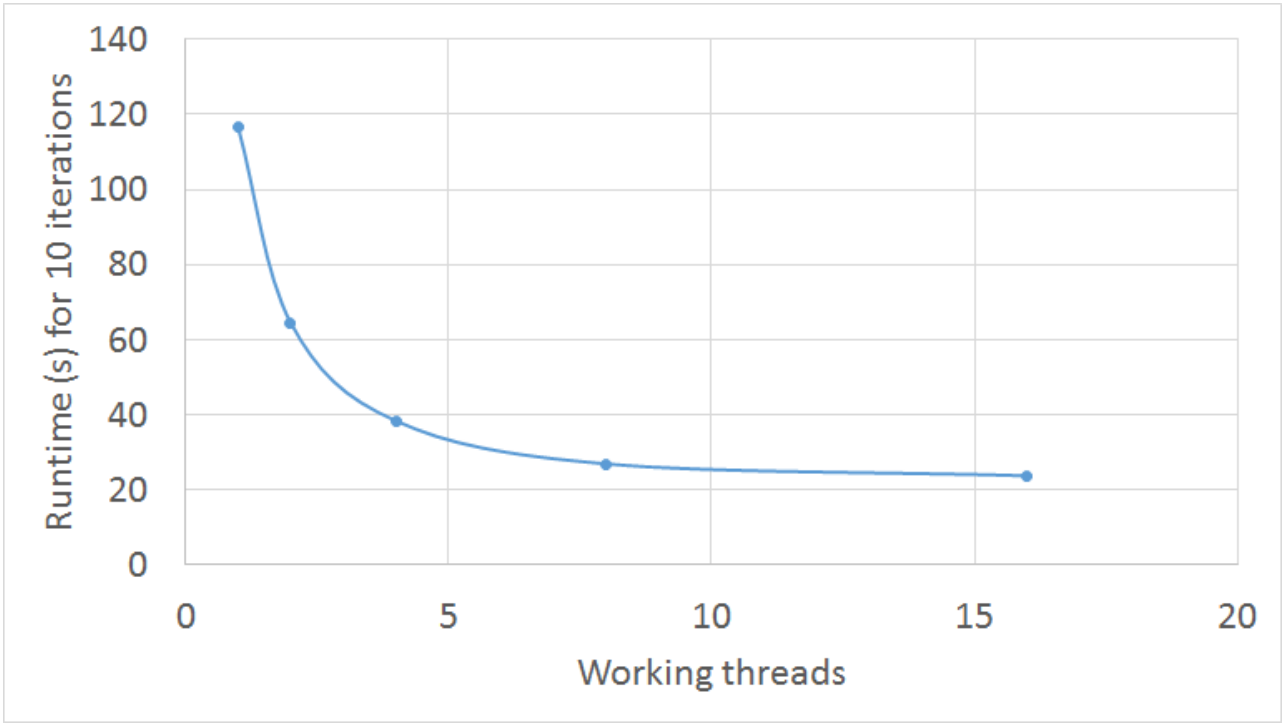
# Pseudocode



```
do {
        // Proposers --> acceptors
        val proposals = proposers.filter( isActive )
                .map( makeProposal ).groupByKey()
                .cache()

        acceptors = acceptors.leftOuterJoin(proposals)
                .mapValues( handleProposal )
                .cache()

        // Acceptors --> proposers
        val responses = acceptors
                .map( makeResponse )
                .cache()

        proposers = proposers.leftOuterJoin(responses)
                .mapValues( handleResponse )
                .cache()

} while ( hasActiveProposers )
```

# Performance Scaling

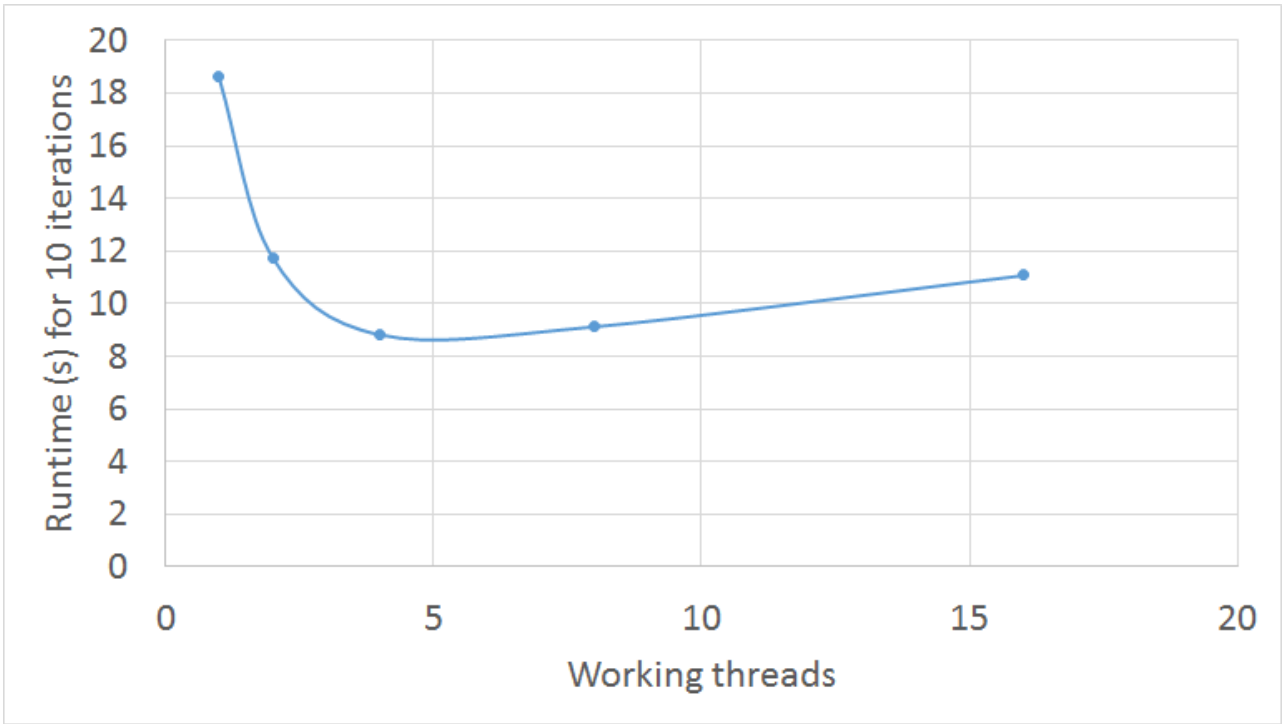





With fixed number of RDD partitions (64 partitions)

# Performance Scaling






With number of RDD partitions equal to number of threads

# Thanks!

https://github.com/gaomy3832/spark-smti