Parallel Apriori Algorithm Implementations

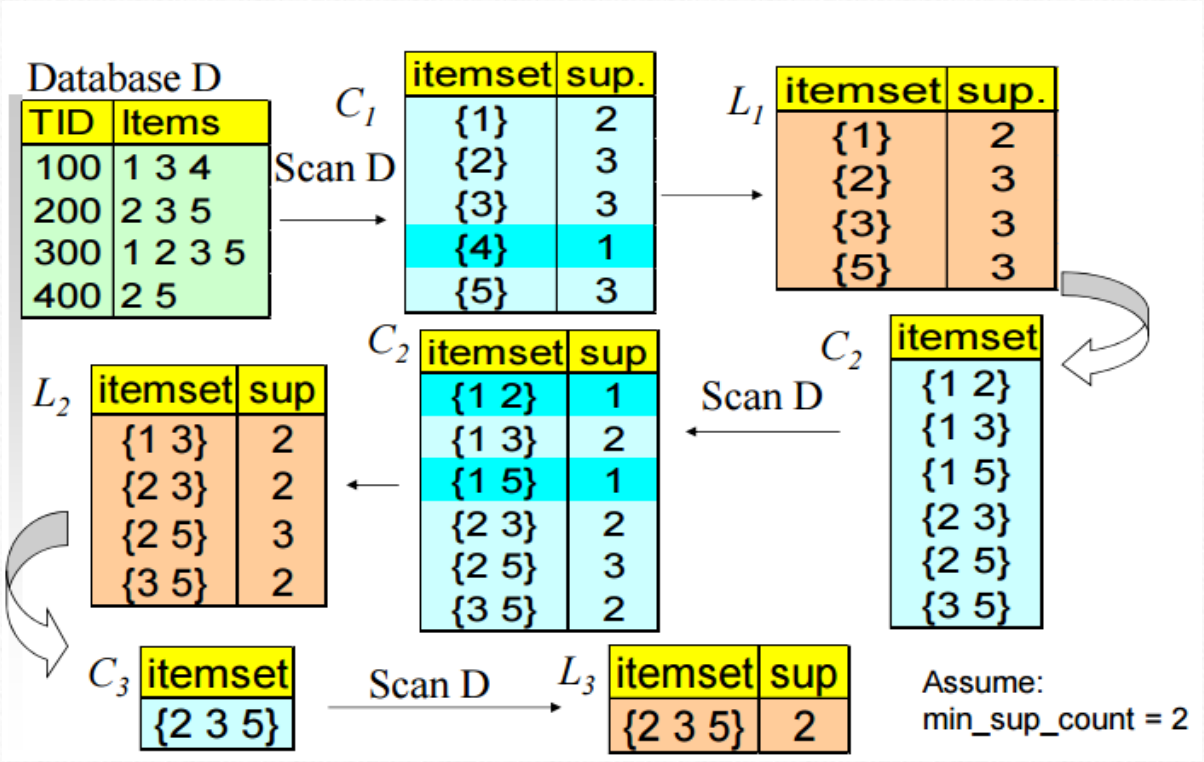
Vassil Halatchev

Department of Electrical Engineering and Computer Science

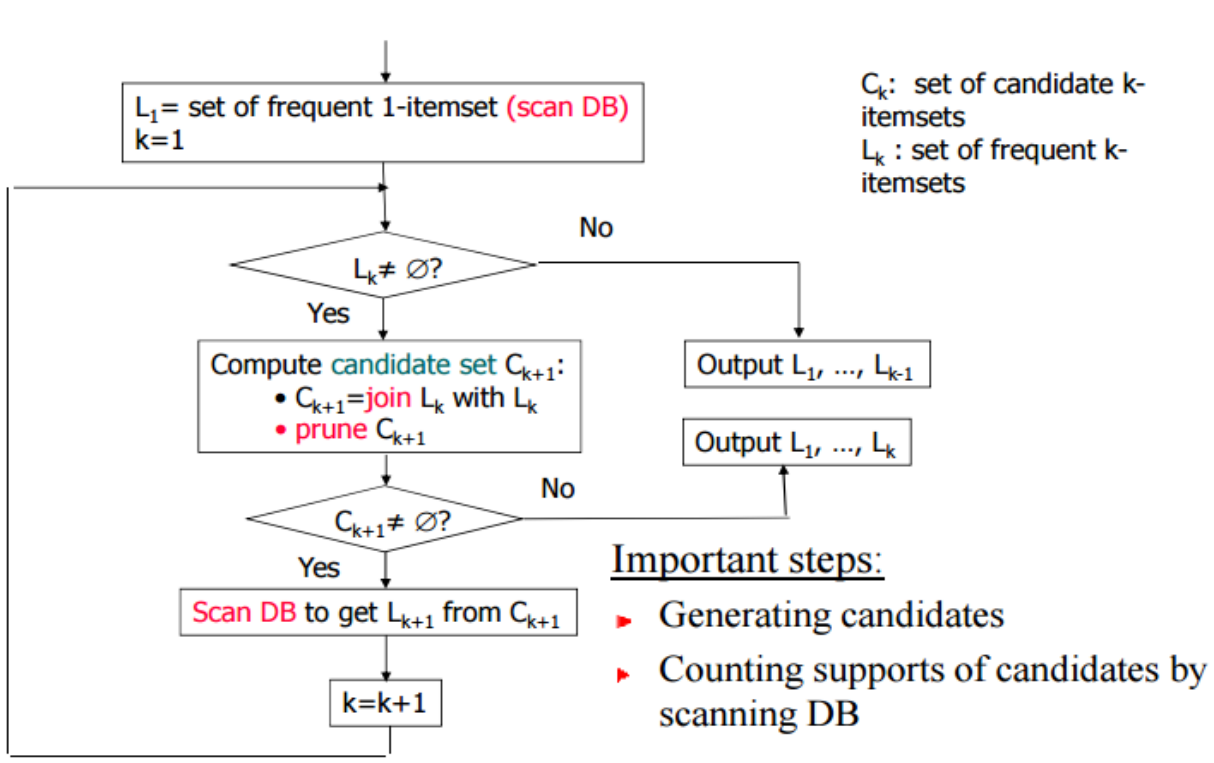
York University, Toronto

November 4, 2015

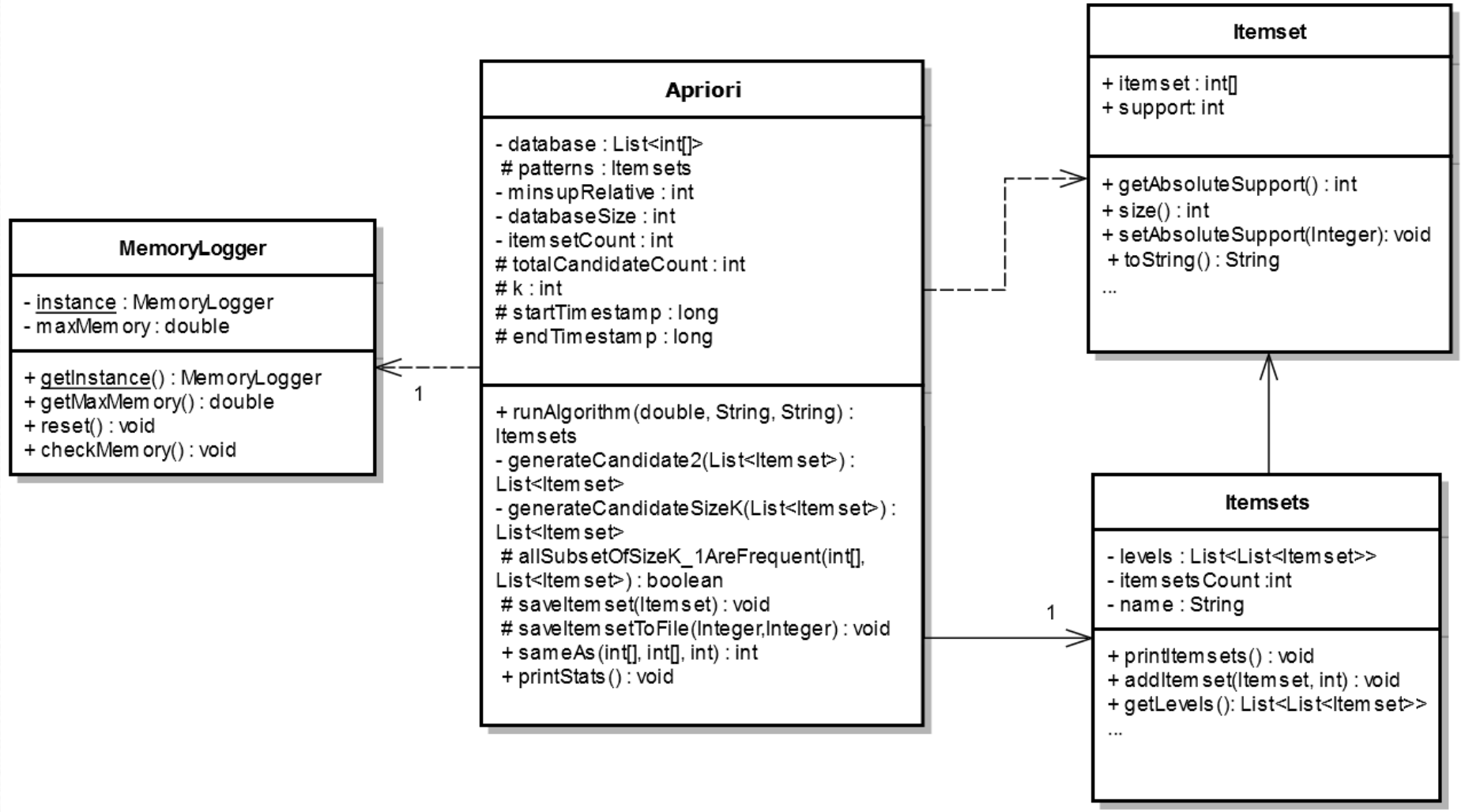
Apriori Algorithm Example



Apriori Algorithm (Flow Chart)



Apriori UML Diagram



Synchronization is Essential

Errors occurred from most to least probable:

1. java.lang.IndexOutOfBoundsException
2. List of frequent itemset is incorrect
3. Support counts are incorrect

Parallel Apriori Algorithm 1: **Count Distribution**

Two different implementations:

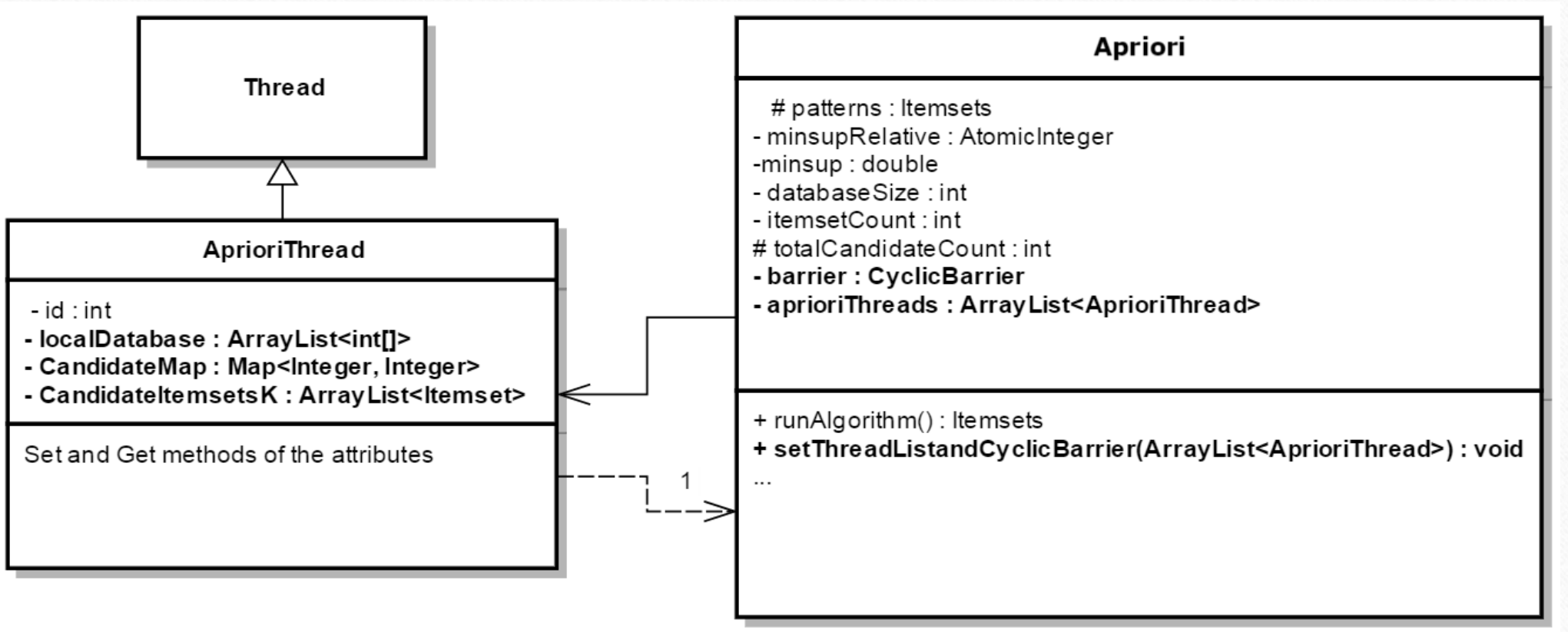
1. **Multiple threads** run on a **single** Apriori instance/object
2. **Each thread** runs on a **new** Apriori instance (one-to-one mapping)

Parallel Apriori Algorithm 1:
Count Distribution
(Implementation 1)

Important points:

- Itemset list preserves it **lexicographical ordering** (e.g. {(1,5), (1,7), (2,3), (3,7,9)})
- Transactions in the database are split **evenly** among the threads
- Only the **first** thread updates the frequent itemset list (i.e. **Itemsets** object).
- The single Apriori object holds a **list of all the threads** that are running on it.
- Synchronization method : **Cyclic Barrier**

Parallel Apriori Algorithm 1:
Count Distribution
(Implementation 1 UML Diagram)

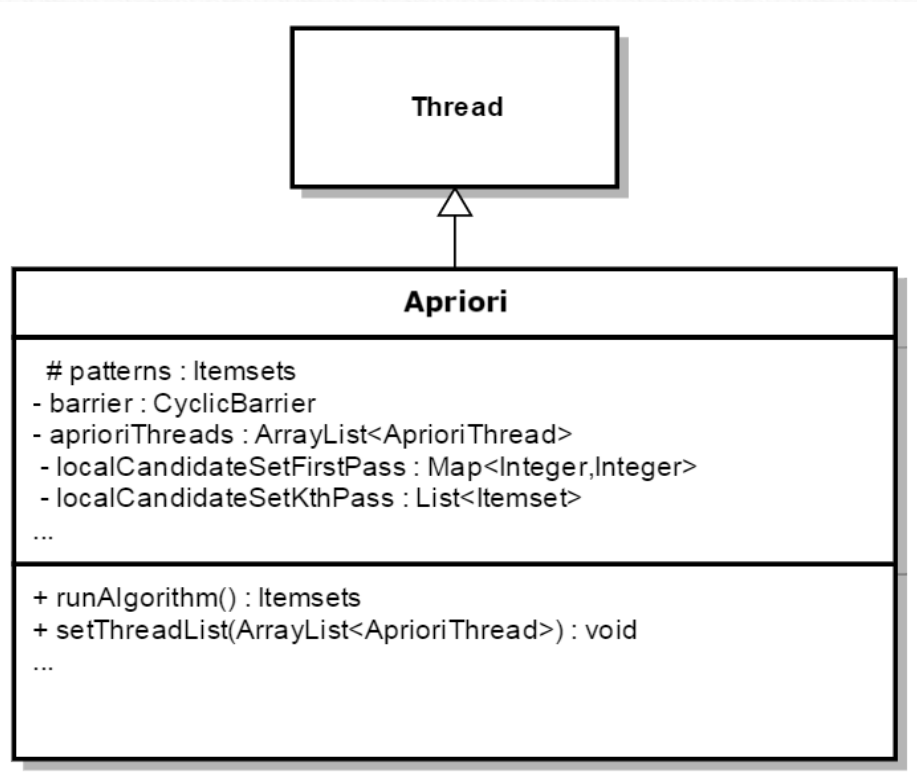


Parallel Apriori Algorithm 1:
Count Distribution
(Implementation 2)

Important points:

- Itemset list preserves it **lexicographical ordering** (e.g. {(1,5), (1,7), (2,3), (3,7,9)})
- Transactions in the database are split evenly among the threads
- Apriori class **extends** Thread class
- Every Apriori object holds a **list of all the threads.**
- Synchronization method : **Cyclic Barrier** (shared object amongst every Apriori object)

Parallel Apriori Algorithm 1:
Count Distribution
(Implementation 2 UML Diagram)



Count Distribution
(Implementation 2 Example)

Minimum Support Count = 2

Step 1: Split database evenly among threads

Transactional Database

TID	Transaction
100	1,3,5,7
200	2,5,6
300	1,2,5
400	3,4,6
500	3,8
600	1,3,6
700	2,5,6

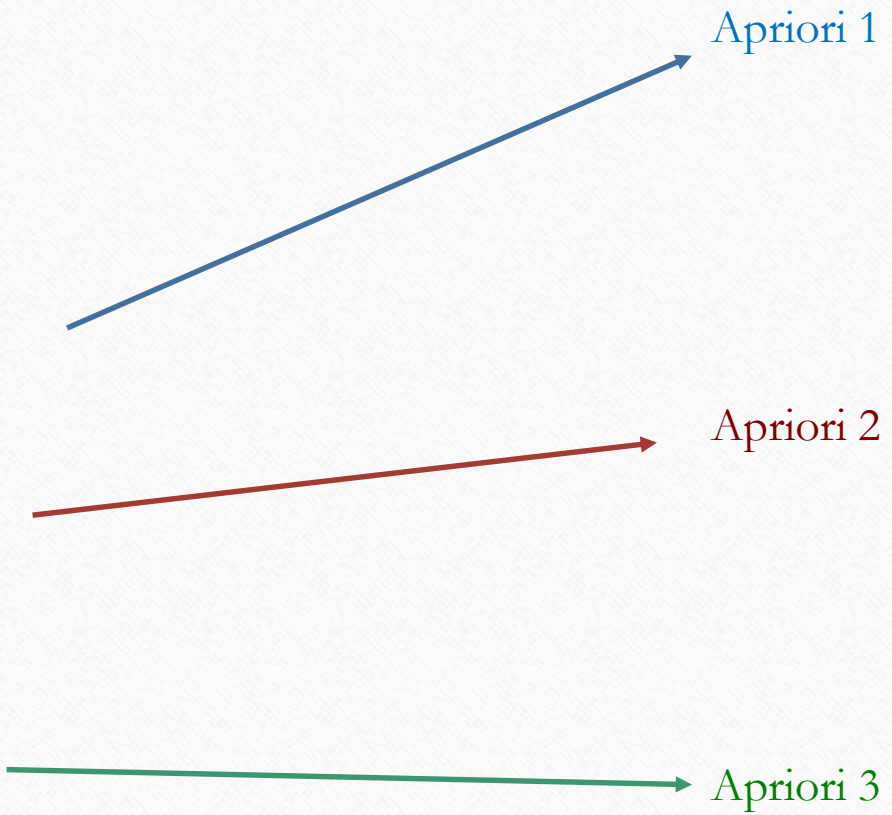
Count Distribution (Implementation 2 Example)

Note: Apriori class **extends** Thread class

Step 1: Split database evenly among threads

Transactional Database

TID	Transaction
100	1,3,5,7
200	2,5,6
300	1,2,5
400	3,4,6
500	3,8
600	1,3,6
700	2,5,6



localDatabase:

TID	Transaction
100	1,3,5,7
200	2,5,6

TID	Transaction
300	1,2,5
400	3,4,6

TID	Transaction
500	3,8
600	1,3,6
700	2,5,6

Pass k = 1

Count Distribution (Implementation 2 Example)

Minimum Support Count = 2

Step 2: Get local C_1 (candidate 1-itemsets) and get local support count.

Apriori 1

TID	Transaction
100	1,3,5,7
200	2,5,6

C_1: (Local)

Itemset	Support
1	1
2	1
3	1
5	2
6	1
7	1

Apriori 2

TID	Transaction
300	1,2,5
400	3,4,6

Itemset	Support
1	1
2	1
3	1
4	1
5	1
6	1

Apriori 3

TID	Transaction
500	3,8
600	1,3,6
700	2,5,6

Itemset	Support
1	1
2	1
3	2
5	1
6	2
8	1

Cyclic Barrier (# of threads)

Pass k = 1

Count Distribution (Implementation 2 Example)

Minimum Support Count = 2

Step 3: Get global C_1 (candidate 1-itemsets) and get global support count.

C_1: (Global)

Apriori 1

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

Apriori 2

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

Apriori 3

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

Pass k = 1

Count Distribution (Implementation 2 Example)

Minimum Support Count = 2

Step 4: - Get global $\mathbf{F_1}$ (frequent 1-itemsets) and get global support count.
- Save $\mathbf{F_1}$.

F_1: (Global)

Apriori 1

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

patterns = {(1),(2),(3),(5),(6)}

Apriori 2

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

patterns = {(1),(2),(3),(5),(6)}

Apriori 3

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

patterns = {(1),(2),(3),(5),(6)}

Pass k = 2

Count Distribution (Implementation 2 Example)

Minimum Support Count = 2

Step 1: Generate C_2 (frequent 2-itemsets), **join & prune,** and get local support count (via scan of local database). **Store C_2 locally.**

C_2: (Local)

Apriori 1

Itemset	Support
(1, 2)	0
(1, 3)	1
(1, 5)	1
(1, 6)	0
(2, 3)	0
(2, 5)	1
(2, 6)	1
(3, 5)	1
(3, 6)	0
(5, 6)	1

Apriori 2

Itemset	Support
(1, 2)	1
(1, 3)	0
(1, 5)	1
(1, 6)	0
(2, 3)	0
(2, 5)	1
(2, 6)	0
(3, 5)	0
(3, 6)	1
(5, 6)	0

Apriori 3

Itemset	Support
(1, 2)	0
(1, 3)	1
(1, 5)	0
(1, 6)	1
(2, 3)	0
(2, 5)	1
(2, 6)	1
(3, 5)	0
(3, 6)	1
(5, 6)	1

Cyclic Barrier (# of threads)

Pass k = 2

Count Distribution (Implementation 2 Example)

Minimum Support Count = 2

Step 2: Get C_2 (candidate 2-itemsets) and get global support count (via scan of local candidate set C_2 of each thread).

C_2: (Global)

Apriori 1

Itemset	Support
(1, 2)	1
(1, 3)	2
(1, 5)	2
(1, 6)	1
(2, 3)	0
(2, 5)	3
(2, 6)	2
(3, 5)	1
(3, 6)	2
(5, 6)	2

Apriori 2

Itemset	Support
(1, 2)	1
(1, 3)	2
(1, 5)	2
(1, 6)	1
(2, 3)	0
(2, 5)	3
(2, 6)	2
(3, 5)	1
(3, 6)	2
(5, 6)	2

Apriori 3

Itemset	Support
(1, 2)	1
(1, 3)	2
(1, 5)	2
(1, 6)	1
(2, 3)	0
(2, 5)	3
(2, 6)	2
(3, 5)	1
(3, 6)	2
(5, 6)	2

Cyclic Barrier (# of threads)

Pass k = 2

Count Distribution (Implementation 2 Example)

Minimum Support Count = 2

Step 3: Get $\mathbf{F_2}$ (frequent 2-itemsets) and get global support count. Save $\mathbf{F_2}$.

F_2: (Global)

Apriori 1

Itemset	Support
(1, 2)	1
(1, 3)	2
(1, 5)	2
(1, 6)	1
(2, 3)	0
(2, 5)	3
(2, 6)	2
(3, 5)	1
(3, 6)	2
(5, 6)	2

patterns = {(1),(2),(3),(5),(6), (1,3),(1,5),(2,5),(2,6), (3,6), (5,6)}

Apriori 2

Itemset	Support
(1, 2)	1
(1, 3)	2
(1, 5)	2
(1, 6)	1
(2, 3)	0
(2, 5)	3
(2, 6)	2
(3, 5)	1
(3, 6)	2
(5, 6)	2

patterns = {(1),(2),(3),(5),(6), (1,3),(1,5),(2,5),(2,6), (3,6), (5,6)}

Apriori 3

Itemset	Support
(1, 2)	1
(1, 3)	2
(1, 5)	2
(1, 6)	1
(2, 3)	0
(2, 5)	3
(2, 6)	2
(3, 5)	1
(3, 6)	2
(5, 6)	2

patterns = {(1),(2),(3),(5),(6), (1,3),(1,5),(2,5),(2,6), (3,6), (5,6)}

Pass k = 3

Count Distribution (Implementation 2 Example)

Minimum Support Count = 2

Step 1: Generate C_3 (frequent 3-itemsets), **join & prune,** and get local support count (via scan of local database). **Store C_3 locally.**

C_3: (Local)

Apriori 1

Itemset	Support
(2, 5, 6)	1

Apriori 2

Itemset	Support
(2, 5, 6)	0

Apriori 3

Itemset	Support
(2, 5, 6)	1

Cyclic Barrier (# of threads)

Pass k = 3

Count Distribution (Implementation 2 Example)

Minimum Support Count = 2

Step 2: Get C_3 (candidate 3-itemsets) and get global support count (via scan of local candidate set C_3 of each thread).

C_3: (Global)

Apriori 1

Itemset	Support
(2, 5, 6)	2

Apriori 2

Itemset	Support
(2, 5, 6)	2

Apriori 3

Itemset	Support
(2, 5, 6)	2

Cyclic Barrier (# of threads)

Pass k = 3

Count Distribution (Implementation 2 Example)

Minimum Support Count = 2

Step 3: Get $\mathbf{F_3}$ (frequent 3-itemset) and get global support count. Save $\mathbf{F_3}$.

F_3: (Global)

Apriori 1

Itemset	Support
(2, 5, 6)	2

patterns = {(1),(2),(3),(5),(6), (1,3),(1,5),(2,5),(2,6), (3,6), (5,6), (2, 5, 6) }

Apriori 2

Itemset	Support
(2, 5, 6)	2

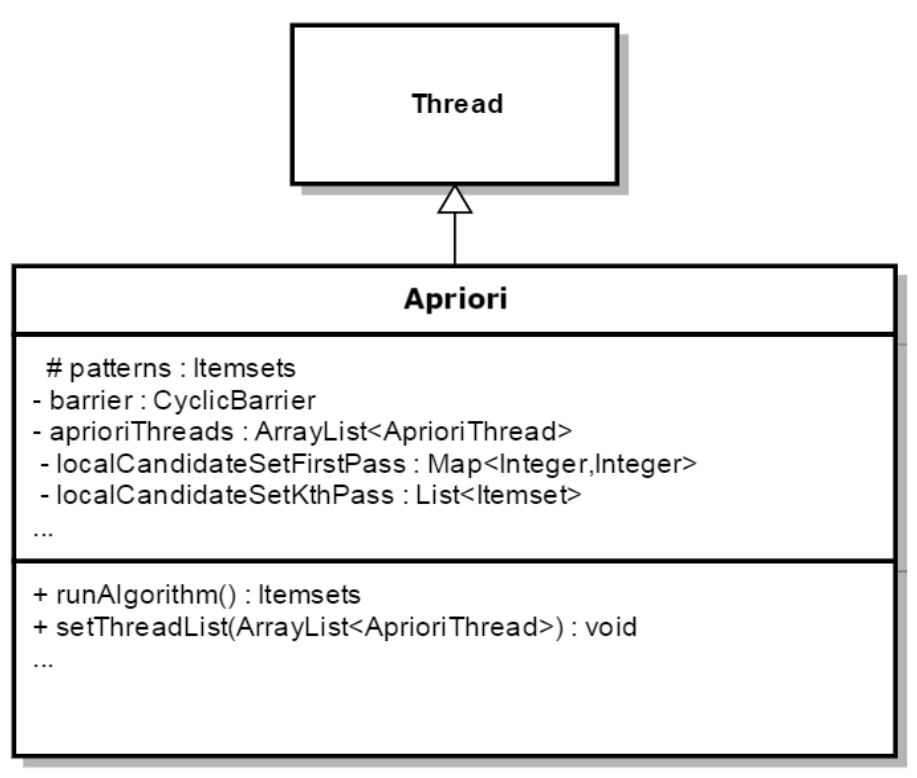
patterns = {(1),(2),(3),(5),(6), (1,3),(1,5),(2,5),(2,6), (3,6), (5,6), (2, 5, 6) }

Apriori 3

Itemset	Support
(2, 5, 6)	2

patterns = {(1),(2),(3),(5),(6), (1,3),(1,5),(2,5),(2,6), (3,6), (5,6), (2, 5, 6) }

Parallel Apriori Algorithm 2:
Data Distribution
UML Diagram



Data Distribution Example

Minimum Support Count = 2

Step 1: Split database evenly among threads

Transactional Database

TID	Transaction
100	1,3,5,7
200	2,5,6
300	1,2,5
400	3,4,6
500	3,8
600	1,3,6
700	2,5,6

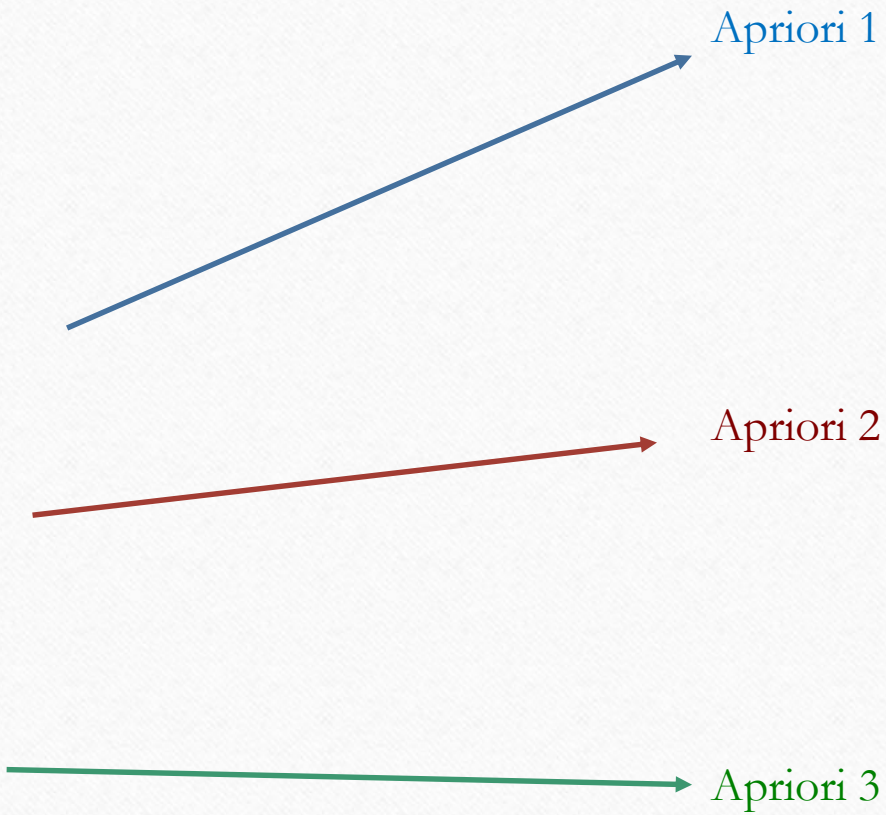
Data Distribution Example

Note: Apriori class **extends** Thread class

Step 1: Split database evenly among threads

Transactional Database

TID	Transaction
100	1,3,5,7
200	2,5,6
300	1,2,5
400	3,4,6
500	3,8
600	1,3,6
700	2,5,6



localDatabase:

TID	Transaction
100	1,3,5,7
200	2,5,6

TID	Transaction
300	1,2,5
400	3,4,6

TID	Transaction
500	3,8
600	1,3,6
700	2,5,6

Pass k = 1

Data Distribution Example

Minimum Support Count = 2

Step 2: Get local C_1 (candidate 1-itemset) and get local support count.

Apriori 1

TID	Transaction
100	1,3,5,7
200	2,5,6

Itemset	Support
1	1
2	1
3	1
5	2
6	1
7	1

Apriori 2

TID	Transaction
300	1,2,5
400	3,4,6

Itemset	Support
1	1
2	1
3	1
4	1
5	1
6	1

Apriori 3

TID	Transaction
500	3,8
600	1,3,6
700	2,5,6

Itemset	Support
1	1
2	1
3	2
5	1
6	2
8	1

C_1: (Local)

Cyclic Barrier (# of threads)

Pass k = 1

Data Distribution Example

Minimum Support Count = 2

Step 3: Get global C_1 (candidate 1-itemset) and get global support count.

C_1: (Global)

Apriori 1

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

Apriori 2

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

Apriori 3

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

Pass k = 1

Data Distribution Example

Minimum Support Count = 2

Step 4: - Get global $\mathbf{F_1}$ (frequent 1-itemset) and get global support count.
- Save $\mathbf{F_1}$.

F_1: (Global)

Apriori 1

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

patterns = {(1),(2),(3),(5),(6)}

Apriori 2

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

patterns = {(1),(2),(3),(5),(6)}

Apriori 3

Itemset	Support
1	3
2	3
3	4
4	1
5	4
6	4
7	1
8	1

patterns = {(1),(2),(3),(5),(6)}

Pass k = 2

Data Distribution Example

Minimum Support Count = 2

Step 1: Generate C_2 (frequent 2-itemsets), **join & prune,** and divide them up in a **round-robin** fashion among the threads.

C_2 = { (1,2), (1,3), (1,5), (1,6), (2,3), (2,5), (2,6), (3,5), (3,6), (5,6) }

Apriori 1

Itemset
(1, 2)
(1, 6)
(2, 6)
(5, 6)

Apriori 2

Itemset
(1, 3)
(2, 3)
(3, 5)

Apriori 3

Itemset
(1, 5)
(2, 5)
(3, 6)

Pass k = 2

Data Distribution Example

Minimum Support Count = 2

Step 2: Each thread scans the local database of **every** other thread to get support counts for its **C_2 partition.**

TID	Transaction
100	1,3,5,7
200	2,5,6

TID	Transaction
300	1,2,5
400	3,4,6

TID	Transaction
500	3,8
600	1,3,6
700	2,5,6

C_2 partitions:

Apriori 1

Itemset	Support
(1, 2)	1
(1, 6)	1
(2, 6)	2
(5, 6)	2

Apriori 2

Itemset	Support
(1, 3)	2
(2, 3)	0
(3, 5)	1

Apriori 3

Itemset	Support
(1, 5)	2
(2, 5)	3
(3, 6)	2

Cyclic Barrier (# of threads)

Pass k = 2

Step 3: Retain only F_2 from each C_2 partition.
Save **F_2 partition locally.**

Data Distribution Example

Minimum Support Count = 2

Apriori 1

TID	Transaction
100	1,3,5,7
200	2,5,6

F_2 partitions:

Itemset	Support
~~(1, 2)~~	~~1~~
~~(1, 6)~~	~~1~~
(2, 6)	2
(5, 6)	2

patterns = {(1),(2),(3),(5),(6) (2,6), (5,6)}

Apriori 2

TID	Transaction
300	1,2,5
400	3,4,6

Itemset	Support
(1, 3)	2
~~(2, 3)~~	~~0~~
~~(3, 5)~~	~~1~~

patterns = {(1),(2),(3),(5),(6) (1,3)}

Apriori 3

TID	Transaction
500	3,8
600	1,3,6
700	2,5,6

Itemset	Support
(1, 5)	2
(2, 5)	3
(3, 6)	2

patterns = {(1),(2),(3),(5),(6) (2,5), (3,6)}

Cyclic Barrier (# of threads)

Pass k = 3

Step 4: Exchange the disjoint **F_2 partition** amongst each thread (we sort at this point).

Data Distribution Example

Minimum Support Count = 2

F_2:

Apriori 1

Itemset	Support
(1, 3)	2
(1, 5)	2
(2, 5)	3
(2, 6)	2
(3, 6)	2
(5, 6)	2

Apriori 2

Itemset	Support
(1, 3)	2
(1, 5)	2
(2, 5)	3
(2, 6)	2
(3, 6)	2
(5, 6)	2

Apriori 3

Itemset	Support
(1, 3)	2
(1, 5)	2
(2, 5)	3
(2, 6)	2
(3, 6)	2
(5, 6)	2

Pass k = 3

Data Distribution Example

Minimum Support Count = 2

Step 1: Generate C_3 (frequent 3-itemsets), **join & prune,** and divide them up in a **round-robin** fashion among the threads.

C_2 = { (2,5,6)}

C_2

partitioned:

Apriori 1

Itemset	Support
(2,5,6)	2

Apriori 2

Itemset	Support

Apriori 3

Itemset	Support

Pass k = 3

Data Distribution Example

Minimum Support Count = 2

Step 2: Each thread scans the local database of **every** other thread to get support counts for its **C_2 partition.**

TID	Transaction
100	1,3,5,7
200	2,5,6

TID	Transaction
300	1,2,5
400	3,4,6

TID	Transaction
500	3,8
600	1,3,6
700	2,5,6

C_2 partitions:

Apriori 1

Itemset	Support
(2, 5, 6)	2

Apriori 2

Itemset	Support

Apriori 3

Itemset	Support

Cyclic Barrier (# of threads)

Pass k = 3

Step 3: Retain only F_2 from each C_2 partition.
Save **F_2 partition locally.**

Data Distribution Example

Minimum Support Count = 2

Apriori 1

TID	Transaction
100	1,3,5,7
200	2,5,6

Itemset	Support
(2,5,6)	2

F_2 partitions:

patterns = {(1),(2),(3),(5),(6) (2,6), (5,6), (2,5,6)}

Apriori 2

TID	Transaction
300	1,2,5
400	3,4,6

Itemset	Support

No change
patterns = {(1),(2),(3),(5),(6) (1,3)}

Apriori 3

TID	Transaction
500	3,8
600	1,3,6
700	2,5,6

Itemset	Support

No change
patterns = {(1),(2),(3),(5),(6) (2,5), (3,6)}

Cyclic Barrier (# of threads)

Parallel Apriori Algorithm 3: **Candidate Distribution**

(Implementation in Progress)

- **Pass k < m:** Use either **Count** or **Data** distribution algorithm
- **Pass k = m:**
  1. Partition L_{k-1} among the N threads such that L_{k-1} sets are **"well balanced".**
  2. Database is repartitioned among threads.
- **Pass k > m:**
  1. Threads proceed **independently** without synchronizing at the end of every pass.
  2. Only dependence among threads is for **pruning the local candidate set.**
  3. Threads don't wait for the complete pruning information, but **opportunistically** starts counting the candidates.

Testing

1. Test Sequential Apriori Implementation extensively
2. Use result obtained from the sequential Implementation to determine accuracy of the Parallel implementations using the same data set for both

Sequential Tests

1. Vary the data set, number of transactions from the data set, average transaction length
2. Use random minimum support values

Use a combination from 1) and 2) to obtain a result **A.**

1. Scan the database-> **2.** get all 1-itemsets, say **n** #'s -> **3.** generate all combinations of **n** to obtain all possible k-itemsets, 2<=k<=**n** -> **4.** Scan database and retain only the frequent itemsets, call this list **B**. ->**5.** Assert **A==B** (this checks:

- The size of A == B
- All itemsets in A are contained in B and all itemsets in B are contained in A
- The support count of the same itemset contained in A and in B is the same.

Concurrent Tests

1. Vary the data set, number of transactions from the data set, average transaction length
2. Use random minimum support values
3. Number of threads
4. The partition proportion of the data set among the threads:

Use the result obtained from a combination on the 1,2,3,4 on a **concurrent implementation** with the result obtained from the combination of 1 and 2 on the **sequential implementation**