PARALLEL SOFTWARE TECHNOLOGIES LABORATORY PSTL

COSC 6397 Big Data Analytics

Master Worker Programming Pattern

Edgar Gabriel
Spring 2015

UNIVERSITY of HOUSTON

PARALLEL SOFTWARE TECHNOLOGIES LABORATORY PSTL

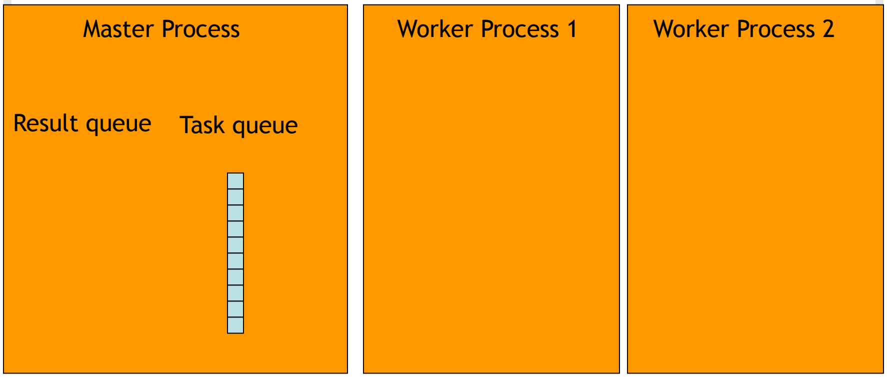
Master-Worker pattern

- General idea: distribute the work among a number of processes
- Two logically different entities:
  - master process: manages assignments to worker processes
  - worker processes executing a task assigned to them by the master
- Communication only occurs between master process and worker processes
  - No direct worker to worker communication possible

UNIVERSITY of HOUSTON

PARALLEL SOFTWARE TECHNOLOGIES LABORATORY PSTL

Master-Worker framework



UNIVERSITY of HOUSTON

PARALLEL SOFTWARE TECHNOLOGIES LABORATORY PSTL

Master process pseudo code

```
for each worker w=0…no. of workers {
        f = FETCH next piece of work
        SEND f to w
}
while ( not done) {
        RECEIVE result from any process
        w = process who sent result
        STORE result
        f = FETCH next piece of work
        if f != no more work left
                SEND f to w
        else
                SEND termination message to w
        }
```

UNIVERSITY of HOUSTON

Master process

- List of things to worry about:
  - Number of workers:
    - Want to maximize performance -> many workers
    - Too many workers: master process can become the bottleneck
  - Work distribution: too little work per worker can lead to large management overhead for the master
  - SEND/RECEIVE
    - can be implemented using any communication paradigm (sockets, MPI, http, files on a shared filesystem,…)
    - More than one message might be required per communication step

Master process

- Support for process failure
  - Master needs to maintain a list of which work item has been assigned to which worker
  - If a worker fails, work is reassigned to new worker
  - Resilience of master can be achieved through reliable back-end storage

- Correctness verification :
  - assign the same work to more than one worker
  - compare results obtained

Worker process pseudo code

```
while ( not done ) {
        RECEIVE work f from master
        If f equals termination signal
                exit
        Else
                result r = execute work on f
                SEND r to master
}
```


Worker process

- Dynamically generating worker instances vs. using a fixed number of workers
- Worker can support multiple functions
  - Additional message might be required to identify the task that the worker needs to execute

1st Example: Word Count

- Application counting the number of occurrences of each word in a given input file
- **Input:** a file with n number of lines
- **Output:** list of words and the number of occurrences
- Two step approach:
  - Step 1:
    - Each worker gets one/a fixed no. of line(s) of the file
    - Worker returns a list of <word, #occurrences> to master.
    - Master writes list of words, #occurrences into a temporary file
  - Step 2: the same word will appear multiple times in the temporary file
    - Sort temporary file
    - Add the number of occurrences of the same word
    - Write word and final number of occurrences to end file

Word count step 1

```
for each worker w=0…no. of workers {
        f = READLINE (file)
        SEND f to w
}
while ( not done) {
        RECEIVE number of elements in the list returned
        allocate temporary buffer to receive list
        RECEIVE list <words, occurrences>
        WRITE list of <words, occurrences> to temporary file
        release temporary buffer
        f = READLINE (file)
        if f != EOF        SEND f to w
        else               SEND termination message to w
}
```

Word count step 2

```
SORT temporary file based on word
<word, occurrence> = READ (temporary file);
currentword = word;
currentcount = occurrence;
while ( not done) {
    <word, occurrence> = READ (temporary file);
     if ( word != currentword ) {
         WRITE (outputfile, currentword, currentcount)
         currentword = word;
         currentcount = 0;
     }
     currentcount += occurrence;
}
```

Alternative design possibilities

- Master searches for word occurrence and adds value received from worker process immediately
  - Avoids temporary file
  - Requires large number of search operations in the file
  - Only final output file is sorted
- Worker process does not return list of <word, occurrence> for each line received
  - Just signals that its ready for a new assignment
  - returns only final list after termination signal

Parallel Software Technologies Laboratory PSTL

Alternative design possibilities (II)

- Using workers for step 2 often also necessary
- Using workers for sorting
  - Requires sending large data volumes
  - Requires direct worker to worker communication

  -> virtually impossible in a classic master-worker setting

  -> Amdahl's law strikes!

- Aggregation of multiple entries for the same word can be distributed to workers

UNIVERSITY of HOUSTON

Parallel Software Technologies Laboratory PSTL

2nd Example: k-means clustering

- An unsupervised clustering algorithm
- "*K*" stands for number of clusters, typically a user input to the algorithm
- Iterative algorithm
  - Might obtain only a local minimum
  - Works only for numerical data
- $x_1, \ldots, x_N$ are data points
- Each data point (vector x_i) will be assigned to exactly one cluster
- Dissimilarity measure: Euclidean distance metric

UNIVERSITY of HOUSTON

Parallel Software Technologies Laboratory PSTL

K-means Algorithm

- $C(i)$: cluster that data point x_i is currently assigned to
- For a current set of cluster means m_k, assign each data point to exactly one cluster:

$$C(i) = \arg\min_{1 \le k \le K} \|x_i - m_k\|^2, \; i = 1, \dots, N$$

- N_k: number of data points in cluster k
- For a given cluster assignment C of the data points, compute the cluster means m_k:

$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, \; k = 1, \dots, K.$$

- Iterate above two steps until convergence

UNIVERSITY of HOUSTON

Parallel Software Technologies Laboratory PSTL

K-means clustering: sequential version

Input :
  $X = \{x_1,\dots,x_k\}$ Instances to be clustered
  K : Number of clusters
Output
  $M = \{m_1,\dots,m_k\}$:cluster centroids
  $m: X \to C$: cluster membership
Algorithm
  Set initial value for M
  while m has changed
    $c^{temp}{}_n = 0,\ n = 1,\dots,\ k$
    $N_n = 0,\ n = 1,\dots,\ k$
    for each $x \in X$
      $C(j)$ = min distance $(x_j,\ c_n),\ n = 1,\dots,\ k$
      $c^{temp}{}_n \mathrel{+}= x_j$
      N_n ++
    end
    recompute C based on c^{temp} and N

K-means clustering

- Initial cluster means
  - Using some other/simpler pre-clustering algorithms
  - Random values
- Random values difficult when using multiple processes
  - Pseudo random number generators
  - Master process creates random numbers and distributes it to workers

Master-worker k-means

```
m= randomly generated initial cluster means
x = data points
for each worker w=0…no. of workers {
        SEND portion of x for worker w
}

Do {
    for each worker w=0…no. of workers {
        SEND m to w
    }
    for each worker w=0…no. of workers {
        RECEIVE C from w
    }
    Calculate N  and ctemp for all cluster
    Recalculate m
} while (m has changed )
```

PARALLEL SOFTWARE TECHNOLOGIES LABORATORY PSTL

Master worker k-means

- Portion of data points x assigned to a worker remains constant -> has to be sent only once
- Need to separate loops to send m and receive C to avoid serialization of the worker processes

- Problem is strongly synchronized
  - All worker processes have to finish before master process can recalculate Nk, ctemp and m
  - Calculating Nk, ctemp and m sequential in this version

  -> Amdahl's law strikes!

UNIVERSITY of HOUSTON

PARALLEL SOFTWARE TECHNOLOGIES LABORATORY PSTL

Summary

- Master worker pattern useful for a number of application scenarios
  - Mostly 'trivially' parallel problems
- Supports dynamic load balancing
  - A more powerful worker will return work faster and get more work assigned than a slow worker
  - Well suited for heterogeneous environments
  - Easy to integrate fault tolerance
- Not well suited for synchronized problems
- Amdahl's law will prevent the model from scaling up to extreme problems

UNIVERSITY of HOUSTON