Introduction to MapReduce

Introduction

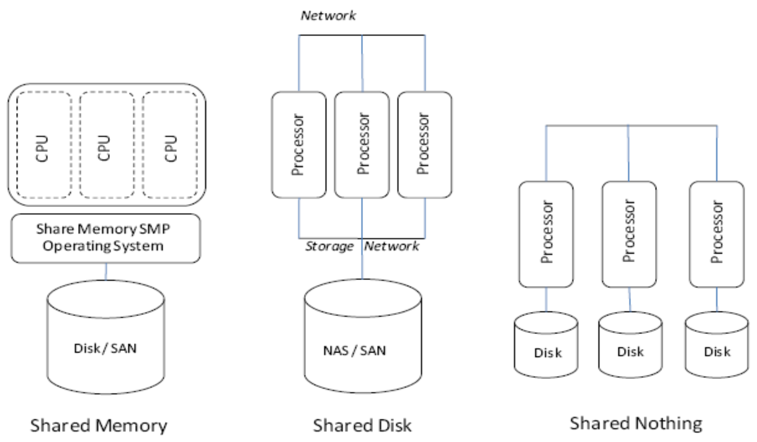
- MapReduce: programming model developed at Google
- Objective:
  - Implement large scale search
  - Text processing on massively scalable web data stored using BigTable and GFS distributed file system
- Designed for processing and generating large volumes of data via massively parallel computations, utilizing tens of thousands of processors at a time
- Fault tolerant: ensure progress of computation even if processors and networks fail
- Example:
  - Hadoop: open source implementation of MapReduce (developed at Yahoo!)
  - Available on pre-packaged AMIs on Amazon EC2 cloud platform

Parallel Computing

- Different models of parallel computing
  - Nature and evolution of multiprocessor computer architecture
  - Shared-memory model
    - Assumes that any processor can access any memory location
    - Unequal latency
  - Distributed-memory model
    - Each processor can access only its own memory and communicates with other processors using message passing
- Parallel computing:
  - Developed for compute intensive scientific tasks
  - Later found application in the database arena
    - Shared-memory
    - Shared-disk
    - Shared-nothing

3 3/19/2023

Parallel Database Architectures



Source: "Enterprise Cloud Computing" by Gautam Shroff

4 3/19/2023

Parallel Database Architectures
Contd…

- Shared memory
  - Suitable for servers with multiple CPUs
  - Memory address space is shared and managed by a symmetric multi-processing (SMP) operating system
  - SMP:
    - Schedules processes in parallel exploiting all the processors
- Shared nothing
  - Cluster of independent servers each with its own disk space
  - Connected by a network
- Shared disk
  - Hybrid architecture
  - Independent server clusters share storage through high-speed network storage viz. NAS (network attached storage) or SAN (storage area network)
  - Clusters are connected to storage via: standard Ethernet, or faster Fiber Channel or Infiniband connections

Parallel Efficiency

- If a task takes time T in uniprocessor system, it should take T/p if executed on p processors
- Inefficiencies introduced in distributed computation due to:
  - Need for synchronization among processors
  - Overheads of message communication between processors
  - Imbalance in the distribution of work to processors
- *Parallel efficiency* of an algorithm is defined as:

$$\epsilon = \frac{T}{p\,T_p}.$$

  - **Scalable** parallel implementation
    - parallel efficiency remains constant as the size of data is increased along with a corresponding increase in processors
    - parallel efficiency increases with the size of data for a fixed number of processors

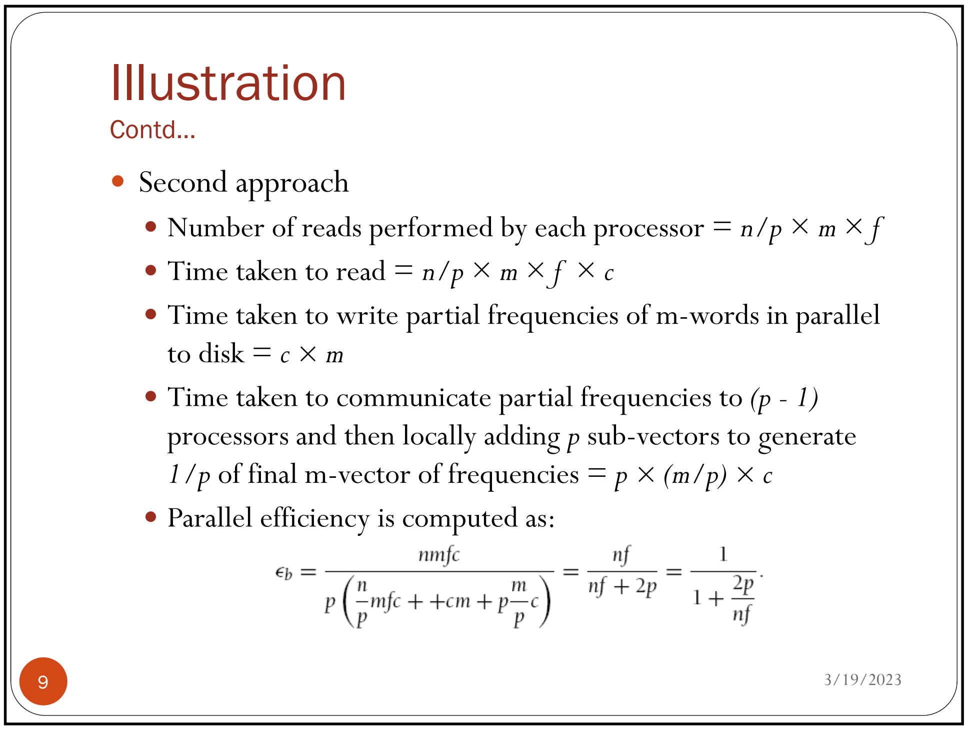
Illustration

- **Problem**: Consider a very large collection of documents, say web pages crawled from the entire Internet. The problem is to determine the frequency (i.e., total number of occurrences) of each word in this collection. Thus, if there are n documents and m distinct words, we wish to determine m frequencies, one for each word.
- Two approaches:
  - Let each processor compute the frequencies for m/p words
  - Let each processor compute the frequencies of m words across n/p documents, followed by all the processors summing their results
- Parallel computing is implemented as a distributed-memory model with a shared disk, so that each processor is able to access any document from disk in parallel with no contention

Illustration Contd...

- Time to read each word from the document = Time to send the word to another processor via inter-process communication = c
- Time to add to a running total of frequencies -> negligible
- Each word occurs f times in a document (on average)
- Time for computing all m frequencies with a single processor = $n \times m \times f \times c$
- First approach:
  - Each processor reads at most $n \times m/p \times f$ times
  - Parallel efficiency is calculated as:
  - Efficiency falls with increasing p
  - *Not scalable*

$$\epsilon_a = \frac{nmfc}{pnmfc} = \frac{1}{p}.$$



Illustration

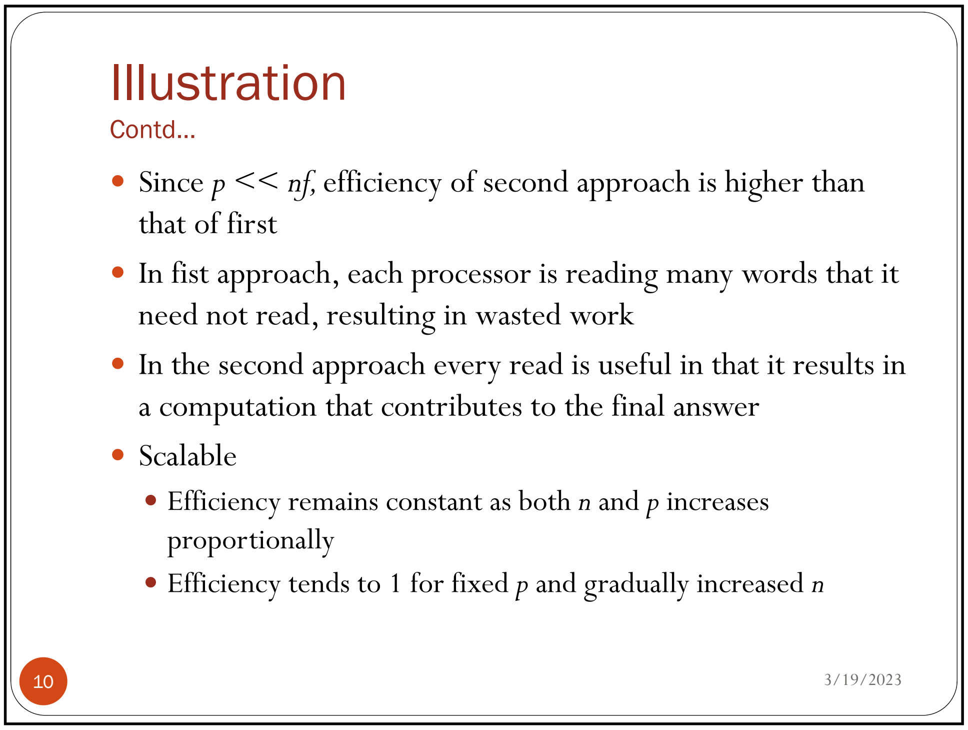
Contd…

- Second approach
  - Number of reads performed by each processor = $n/p \times m \times f$
  - Time taken to read = $n/p \times m \times f \times c$
  - Time taken to write partial frequencies of m-words in parallel to disk = $c \times m$
  - Time taken to communicate partial frequencies to $(p - 1)$ processors and then locally adding p sub-vectors to generate $1/p$ of final m-vector of frequencies = $p \times (m/p) \times c$
  - Parallel efficiency is computed as:

$$\epsilon_b = \frac{nmfc}{p\left(\frac{n}{p}mfc + +cm + p\frac{m}{p}c\right)} = \frac{nf}{nf + 2p} = \frac{1}{1 + \frac{2p}{nf}}.$$

9

3/19/2023



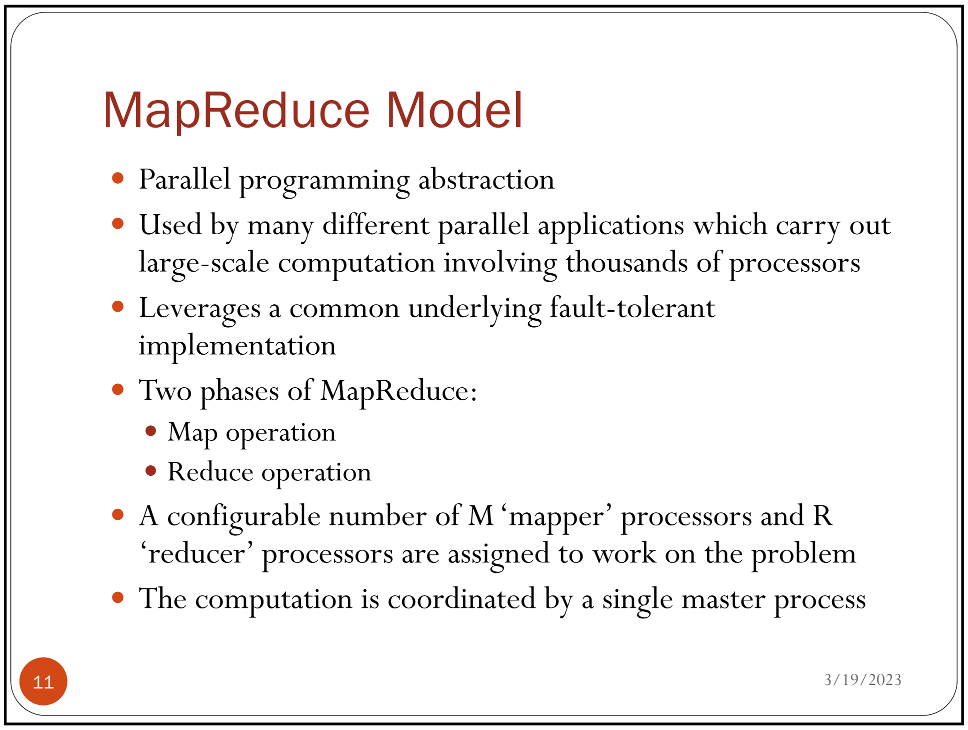
Illustration

Contd…

- Since $p << nf$, efficiency of second approach is higher than that of first
- In fist approach, each processor is reading many words that it need not read, resulting in wasted work
- In the second approach every read is useful in that it results in a computation that contributes to the final answer
- Scalable
  - Efficiency remains constant as both n and p increases proportionally
  - Efficiency tends to 1 for fixed p and gradually increased n

10

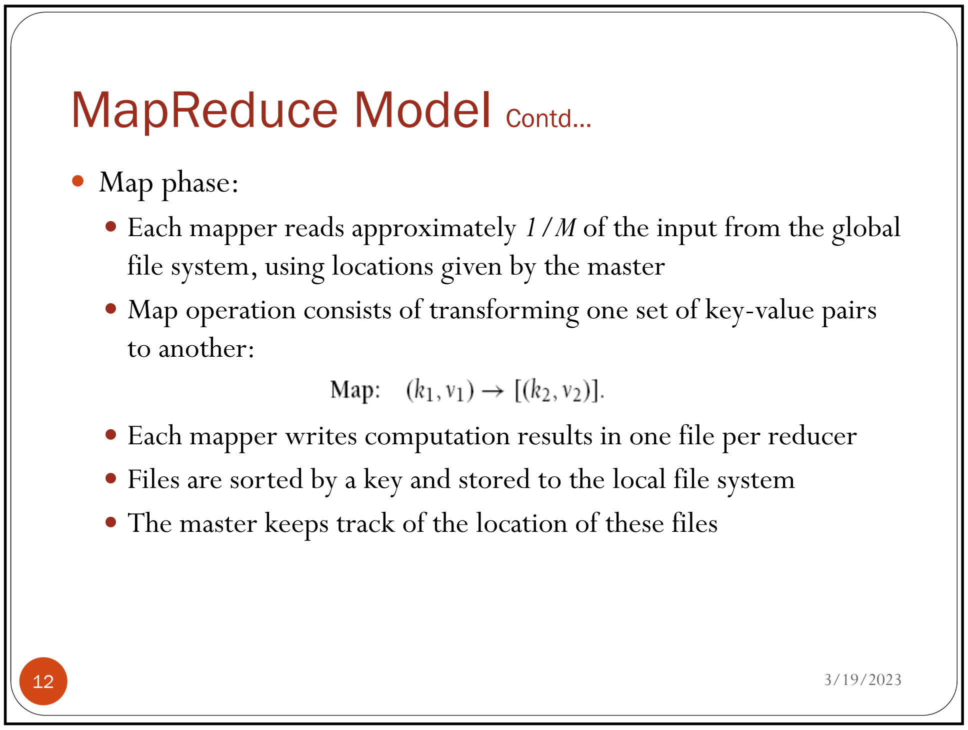
3/19/2023



MapReduce Model

- Parallel programming abstraction
- Used by many different parallel applications which carry out large-scale computation involving thousands of processors
- Leverages a common underlying fault-tolerant implementation
- Two phases of MapReduce:
  - Map operation
  - Reduce operation
- A configurable number of M 'mapper' processors and R 'reducer' processors are assigned to work on the problem
- The computation is coordinated by a single master process

11 3/19/2023



MapReduce Model Contd...

- Map phase:
  - Each mapper reads approximately *1 / M* of the input from the global file system, using locations given by the master
  - Map operation consists of transforming one set of key-value pairs to another:

$$\text{Map:} \quad (k_1, v_1) \rightarrow [(k_2, v_2)].$$

  - Each mapper writes computation results in one file per reducer
  - Files are sorted by a key and stored to the local file system
  - The master keeps track of the location of these files

12 3/19/2023

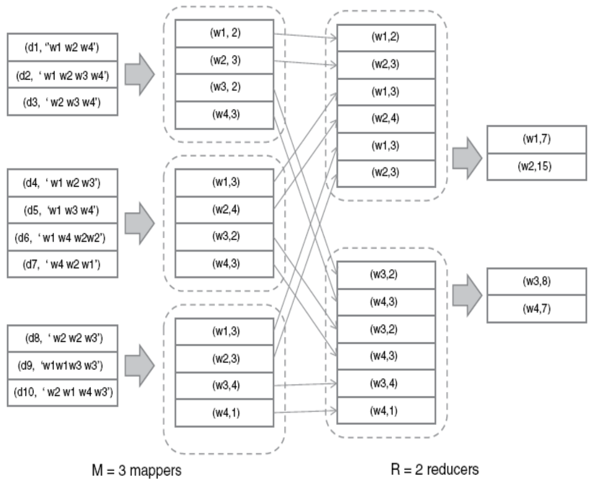
MapReduce Model Contd…

- Reduce phase:
  - The master informs the reducers where the partial computations have been stored on local files of respective mappers
  - Reducers make remote procedure call requests to the mappers to fetch the files
  - Each reducer groups the results of the map step using the same key and performs a function *f* on the list of values that correspond to these key value:
  - Final results ar **Reduce:** $(k_2, [v_2]) \rightarrow (k_2, f([v_2]))$. system

13

3/19/2023

MapReduce: Example



- 3 mappers; 2 reducers
- Map function:

  $(d_k, [w_1 \ldots w_n]) \rightarrow [(w_i, c_i)]$.
- Reduce function:

  $$(w_i, [c_i]) \rightarrow \left(w_i, \sum_i c_i\right)$$

14

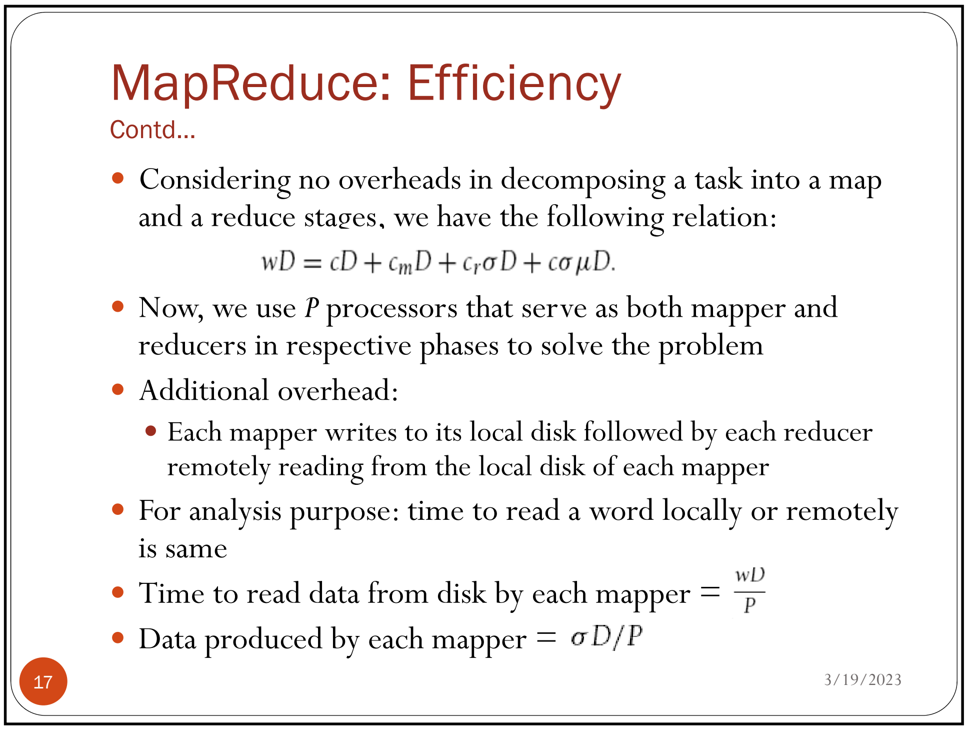
3/19/2023

MapReduce: Fault Tolerance

- Heartbeat communication
  - Updates are exchanged regarding the status of tasks assigned to workers
  - Communication exists, but no progress: master duplicate those tasks and assigns to processors who have already completed
- If a mapper fails, the master reassigns the key-range designated to it to another working node for re-execution
  - Re-execution is required as the partial computations are written into local files, rather than GFS file system
- If a reducer fails, only the remaining tasks are reassigned to another node, since the completed tasks are already written back into GFS

MapReduce: Efficiency

- General computation task on a volume of data *D*
- Takes *wD* time on a uniprocessor (time to read data from disk + performing computation + time to write back to disk)
- Time to read/write one word from/to disk = *c*
- Now, the computational task is decomposed into map and reduce stages as follows:
  - Map stage:
    - Mapping time = $c_m D$
    - Data produced as output = σD
  - Reduce stage:
    - Reducing time = $c_r \sigma D$
    - Data produced as output = $\sigma \mu D$



MapReduce: Efficiency

Contd...

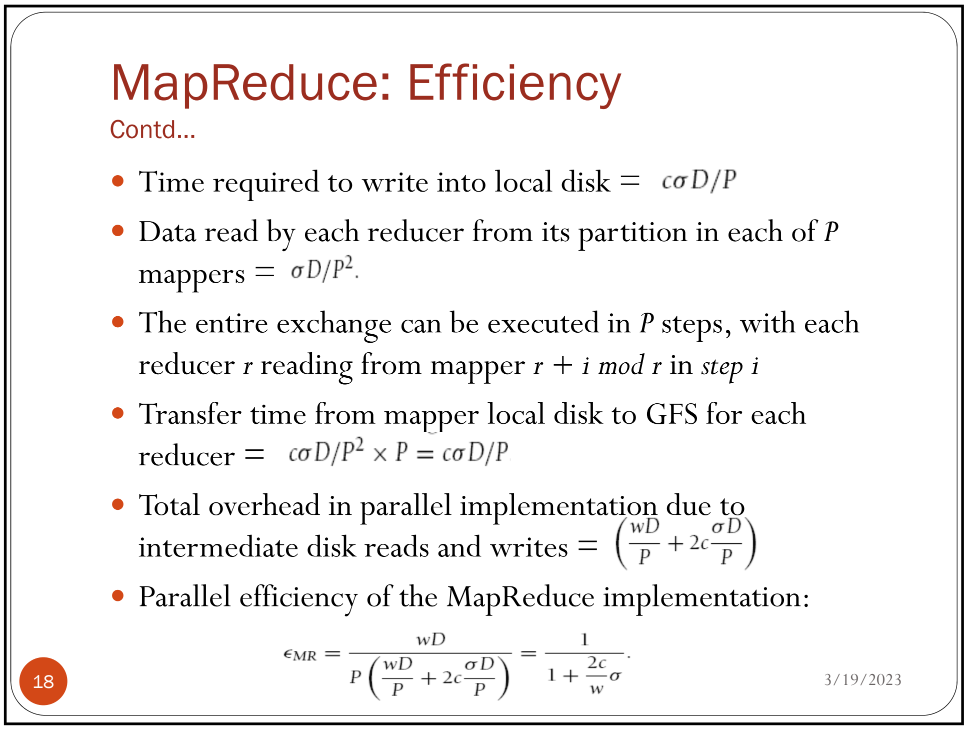
- Considering no overheads in decomposing a task into a map and a reduce stages, we have the following relation:

$$wD = cD + c_m D + c_r \sigma D + c\sigma \mu D.$$

- Now, we use P processors that serve as both mapper and reducers in respective phases to solve the problem
- Additional overhead:
  - Each mapper writes to its local disk followed by each reducer remotely reading from the local disk of each mapper
- For analysis purpose: time to read a word locally or remotely is same
- Time to read data from disk by each mapper $= \frac{wD}{P}$
- Data produced by each mapper $= \sigma D/P$

17

3/19/2023



MapReduce: Efficiency

Contd...

- Time required to write into local disk $= c\sigma D/P$
- Data read by each reducer from its partition in each of P mappers $= \sigma D/P^2$.
- The entire exchange can be executed in P steps, with each reducer r reading from mapper $r + i \ mod \ r$ in $step\ i$
- Transfer time from mapper local disk to GFS for each reducer $= c\sigma D/P^2 \times P = c\sigma D/P$
- Total overhead in parallel implementation due to intermediate disk reads and writes $= \left(\frac{wD}{P} + 2c\frac{\sigma D}{P}\right)$
- Parallel efficiency of the MapReduce implementation:

$$\epsilon_{MR} = \frac{wD}{P\left(\frac{wD}{P} + 2c\frac{\sigma D}{P}\right)} = \frac{1}{1 + \frac{2c}{w}\sigma}.$$

18

3/19/2023

MapReduce: Applications

- Indexing a large collection of documents
  - Important aspect in web search as well as handling structured data
  - The map task consists of emitting a word-document/record-id pair for each word:

$$(d_k, [w_1 \ldots w_n]) \rightarrow [(w_i, d_k)].$$

  - The reduce step groups the pairs by word and creates an index entry for each word:

$$[(w_i, d_k)] \rightarrow (w_i, [d_{i_1} \ldots d_{i_m}]).$$

- Relational operations using MapReduce
  - Execute SQL statements (relational joins/group by) on large data sets
  - Advantages over parallel database
    - Large scale
    - Fault-tolerance

THANK YOU