Distributed Computation of $\pi$ with Apache Hadoop

Tsz-Wo Sze
Yahoo! Cloud Computing
Apache Hadoop PMC Member

Mapred’2010
Dec 1
Agenda

- Introduction
- A New World Record
- How to Compute The $n^{\text{th}}$ Bits of $\pi$?
- Computing $\pi$ with Hadoop
Agenda

● Introduction

● A New World Record

● How to Compute The $n^{th}$ Bits of $\pi$?

● Computing $\pi$ with Hadoop
What is $\pi$?

$\pi$ is a mathematical constant such that, for any circle,

$$\pi = \frac{\text{circumference}}{\text{diameter}} = \frac{C}{d}.$$
What is $\pi$?

- $\pi$ is a mathematical constant such that, for any circle,

$$\pi = \frac{\text{circumference}}{\text{diameter}} = \frac{C}{d}.$$

- We have $\pi = 3.244$
What is $\pi$?

- $\pi$ is a mathematical constant such that, for any circle,
  \[
  \pi = \frac{\text{circumference}}{\text{diameter}} = \frac{C}{d}.
  \]

- We have $\pi = 3.244$ (in hexadecimal 🌔)
Decimal, Hexadecimal & Binary

▶ Representing $\pi$ in different bases

\[ \pi = 3.1415926535 \ 8979323846 \ 2643383279 \ldots \]
\[ = 3.243F_{\text{hex}} \]
\[ = 11.00100100 \ 00111111 \ 01101010 \ldots \]

▶ Bit position is counted after the radix point.

▶ e.g., the eight bits starting at the ninth bit position are 00111111 in binary or 3F in hexadecimal.
Two Types of Challenges

▶ Computing the first $n$ decimal digits of $\pi$

$$\pi = 3.1415926535\ 8979323846\ 2643383279\ldots$$

▶ Computing only the $n^{\text{th}}$ bits of $\pi$

$$\pi = 11.00100100\ 00111111\ 01101010\ 10001000\ldots$$

We will focus on the second challenge in this talk.
Previous Results

▸ Fabrice Bellard (1997)

- *Farthest bit position*: 1,000,000,000,151
  \( (= 10^{12} + 151) \)
- *Precision*: 152 bits
- *Machines*: 20 workstations
- *Duration*: 12 days
- *CPU time*: 220 days
- *Verification*: 180 days CPU time
Previous Results

- **PiHex (2000)**
  - Farthest bit position: $1,000,000,000,000,060$  
    \[= 10^{15} + 60\]
  - Precision: 64 bits
  - Machines: Idle slices of 1734 machines
    An ‘average’ computer has a 450 MHz CPU
  - Duration: **736 days (\(>2\) years)**
  - CPU time: 137 years
  - Verification: ???

It is not clear if they have verified their results.
Agenda

• Introduction

• A New World Record

• How to Compute The $n^{\text{th}}$ Bits of $\pi$?

• Computing $\pi$ with Hadoop
A New World Record

▶ Bit values (in hexadecimal)

0E6C1294 AED40403 F56D2D76 4026265B
CA98511D 0FCFFAA1 0F4D28B1 BB5392B8
A New World Record

- Bit values (in hexadecimal)
  
  0E6C1294  AED40403  F56D2D76  4026265B  
  CA98511D  0FCFFAA1  0F4D28B1  BB5392B8

  (256 bits)

★ The first bit position: 1,999,999,999,999,997 (= 2 \cdot 10^{15} - 3)

★ The last bit position: 2,000,000,000,000,252 (= 2 \cdot 10^{15} + 252)

★ The two quadrillionth (2 \cdot 10^{15}th) bit is 0.
A New World Record

Yahoo! Cloud Computing (July 2010)

- **Farthest bit position**: 2,000,000,000,000,252
- **Precision**: 256 bits
- **Machines**: Idle slices of 1000-node clusters
  - Each node has two quad-core 1.8-2.5 GHz CPUs
- **Duration**: 23 days
- **CPU time**: 503 years
- **Verification**: 582 years CPU time
Comparing with PiHex

<table>
<thead>
<tr>
<th></th>
<th>PiHex</th>
<th>Our Computations</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position:</td>
<td>around $10^{15}$</td>
<td>around $2 \cdot 10^{15}$</td>
<td>1:2</td>
</tr>
<tr>
<td>Precision:</td>
<td>64 bits</td>
<td>256 bits</td>
<td>1:4</td>
</tr>
<tr>
<td>Duration:</td>
<td>736 days</td>
<td>23 days</td>
<td>32:1</td>
</tr>
</tbody>
</table>

Note that our hardware is 10 years more advanced than the ones used by PiHex.
Pi record smashed as team finds two-quadrillionth digit

http://www.bbc.co.uk/news/technology-11313194

A researcher has calculated the 2,000,000,000,000,000th digit of the mathematical constant pi - and a few digits either side of it.

Nicholas Sze, of tech firm Yahoo, said that when pi is expressed in binary, the two quadrillionth "bit" is 0.

Mr Sze used Yahoo's Hadoop cloud computing technology to more than double the previous record.

The formula turns an infinite sum into a more manageable calculation of single terms.
New pi record exploits Yahoo’s computers


New pi record exploits Yahoo’s computers

18:56 17 September 2010 by Davd Shiga

A Yahoo researcher has made a record-breaking calculation of the digits of pi using the company’s computers. The feat comes hot on the heels of a breakthrough Rubik’s cube result that used Google’s computers. Together, the results highlight the growing power of internet search giants to make mathematical breakthroughs.

One way to show off computing power is to calculate pi to as many digits as possible, creating a string that starts with 3.14 and continues to the nth digit. The more digits one wants, the more computations it takes.

But it is also possible to skip ahead to the nth digit without calculating the preceding ones — for example, determining that the 10th digit is 3, without having to find the first 9 digits: 3.14159265. This is another way of testing computing power, since more computations are required to find higher values of n.

Now, Tsz-Wo Sze, a computer scientist at Yahoo in Sunnyvale, California, has used the company’s computers to calculate the most distant digits yet.
Other News Coverage

New Pi Record Exploits Yahoo’s Computers


The Yahoo! boffin scores pi’s two quadrillionth bit

http://www.theregister.co.uk/2010/09/16/pi_record_at_yahoo

Pi calculation more than doubles old record

http://www.radionz.co.nz/news/world/57128/pi-calculation-more-than-doubles-old-record

Hadoop used to calculate Pi’s two quadrillionth bit

http://www.zdnet.co.uk/blogs/mapping-babel-10017967/hadoop-used-to-calculate-pis-two-quadrillionth-bit
Yahoo! researcher breaks Pi record in finding the two-quadrillionth digit

Nicholas Sze of Yahoo Finds Two-Quadrillionth Digit of Pi

The 2,000,000,000,000,000,000th digit of the mathematical constant pi discovered

Researcher Shatters Pi Record by Finding Two-Quadrillionth Digit
A bigger slice of pi

2 Quadrillionth digit of PI is found: Scientist celebration in worldwide Pandemonium

And the number is...0
http://www.hexus.net/content/item.php?item=26505

Pi Record Smashed as Team Finds Two-Quadrillionth Digit
Yahoo Engineer Calculates Two Quadrillionth Bit Of Pi


A Cloud Computing Milestone: Yahoo! Reaches the 2 Quadrillionth Bit of Pi


Yahoo researcher Nicolas Sze determines the 2,000,000,000,000,000th digit of the mathematical constant pi

Other Results

► We also have computed
  • the first billion bits, and
  • around the positions $n = 10^m$ for $m \leq 15$.

► The first billion ($10^9$) bits
  • Arbitrary precision arithmetic

<table>
<thead>
<tr>
<th>Starting Bit Position</th>
<th>Precision (bits)</th>
<th>Time Used</th>
<th>CPU Time</th>
<th>Date Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>800,001,000</td>
<td>10 days</td>
<td>19 years</td>
<td>June 23, 2010</td>
</tr>
<tr>
<td>800,000,001</td>
<td>200,001,000</td>
<td>3 days</td>
<td>8 years</td>
<td>June 22, 2010</td>
</tr>
</tbody>
</table>
Ten & Hundred Trillion

- $n = 10^{13}, 10^{14}$

- It appears that both results are new.
- $n = 10^{13}$

- Verified with Alexander Yee
- 5 trillion decimal digits (August 2010)
- $\approx 1.66 \cdot 10^{13}$ bits
- These two results agree 😊
One Quadrillion

$n = 10^{15}$

The result is similar to the one obtained by PiHex except:

- the chosen starting positions are slightly different
- our result has higher precision (228-bit vs 64-bit)

The overlapped bits of these two results agree. 😊
Agenda

- Introduction
- A New World Record
- How to Compute The $n^{th}$ Bits of $\pi$?
- Computing $\pi$ with Hadoop
The BBP Formula

Bailey, Borwein and Plouffe (1996)

\[ \pi = \sum_{k=0}^{\infty} \frac{1}{2^{4k}} \left( \frac{4}{8k + 1} - \frac{2}{8k + 4} - \frac{1}{8k + 5} - \frac{1}{8k + 6} \right) \]

The above equation is called the BBP formula.

This remarkable discovery leads to the first digit-extraction algorithm for \( \pi \) in base 2.

- allow computing the \( n^{th} \) bits without computing the earlier bits
Another BBP-type Formula

Bellard (1997)

\[
\pi = \sum_{k=0}^{\infty} \frac{(-1)^k}{2^{10k}} \left( \frac{2^2}{10k + 1} - \frac{1}{10k + 3} - \frac{2^{-4}}{10k + 5} - \frac{2^{-4}}{10k + 7} + \frac{2^{-6}}{10k + 9} - \frac{2^{-1}}{4k + 1} - \frac{2^{-6}}{4k + 3} \right)
\]

43% faster than the BBP formula
Computing The \((n + 1)^\text{th}\) Bits of \(\pi\)

In order to obtain the \((n + 1)^\text{th}\) bits,

- multiply \(\pi\) by \(2^n\), and
- take the fraction part,

\[
\{2^n\pi\}, \quad \text{where} \quad \{x\} \overset{\text{def}}{=} x - \lfloor x \rfloor.
\]

For examples,

\[
\{3.14\} = 0.14 \quad \quad \text{(fraction part)}
\]
\[
\lfloor 3.14 \rfloor = 3 \quad \quad \text{(integer part)}
\]
Example

Suppose \( n + 1 = 9 \).

\[
\pi = 11.00100100{\overline{00111111}}\ldots
\]

\[
\{2^n \pi\} = \{2^8 \pi\} = \{11\ 00100100.00111111\ldots\} = .00111111\ldots
\]
The BBP Algorithm

Using BBP formula

\[ \pi = \sum_{k=0}^{\infty} \frac{1}{2^{4k}} \left( \frac{4}{8k + 1} - \frac{2}{8k + 4} - \frac{1}{8k + 5} - \frac{1}{8k + 6} \right), \]

we have

\[ \{2^n \pi\} = \left\{ \sum_{k=0}^{\infty} \frac{2^{n+2-4k}}{8k + 1} - \sum_{k=0}^{\infty} \frac{2^{n-1-4k}}{2k + 1} - \sum_{k=0}^{\infty} \frac{2^{n-4k}}{8k + 5} - \sum_{k=0}^{\infty} \frac{2^{n-1-4k}}{4k + 3} \right\}. \]
Drop The Integer Part Earlier

\[ \{2^n \pi\} = \left\{ \sum_{k=0}^{\infty} \frac{2^{n+2-4k}}{8k + 1} \right\} - \left\{ \sum_{k=0}^{\infty} \frac{2^{n-1-4k}}{2k + 1} \right\} \]

\[ - \left\{ \sum_{k=0}^{\infty} \frac{2^{n-4k}}{8k + 5} \right\} - \left\{ \sum_{k=0}^{\infty} \frac{2^{n-1-4k}}{4k + 3} \right\} \]
\[
\{2^n \pi\} = \left\{ \sum_{k=0}^{\infty} \frac{2^n + 2 - 4k}{8k + 1} \right\} - \left\{ \sum_{k=0}^{\infty} \frac{2^n - 1 - 4k}{2k + 1} \right\} \\
- \left\{ \sum_{k=0}^{\infty} \frac{2^n - 4k}{8k + 5} \right\} - \left\{ \sum_{k=0}^{\infty} \frac{2^n - 1 - 4k}{4k + 3} \right\}
\]
Split The Summations

For each sum, write

\[
\left\{ \sum_{k=0}^{\infty} \left\{ \frac{2^{n+x-4k}}{yk + z} \right\} \right\} = \left\{ \sum_{\substack{n+x-4k > 0 \\k \geq 0}} A_k + \sum_{\substack{n+x-4k \leq 0 \\k \geq 0}} B_k \right\},
\]

where

\[
A_k \overset{\text{def}}{=} \frac{2^{n+x-4k} \mod (yk + z)}{yk + z},
\]

\[
B_k \overset{\text{def}}{=} \frac{1}{2^{4k-n-x}(yk + z)}.
\]
Split The Summations’

For each sum, write

\[
\left\{ \sum_{k=0}^{\infty} \left\{ \frac{2^{n+x-4k}}{y^k + z} \right\} \right\} = \left\{ \sum_{n+x-4k > 0} A_k + \sum_{n+x-4k \leq 0} B_k \right\},
\]

where

\[
A_k \overset{\text{def}}{=} \frac{2^{n+x-4k} \mod (y^k + z)}{y^k + z},
\]

\[
B_k \overset{\text{def}}{=} \frac{1}{2^{4k-n-x}(y^k + z)}.
\]
Split The Summations 

For each sum, write

\[
\left\{ \sum_{k=0}^{\infty} \left\{ \frac{2^{n+x-4k}}{yk + z} \right\} \right\} = \left\{ \begin{array}{ll}
\sum_{n+x-4k > 0}^{k \geq 0} A_k + \sum_{n+x-4k \leq 0}^{k \geq 0} B_k \\
\end{array} \right\}
\]

where

\[
A_k \overset{\text{def}}{=} \frac{2^{n+x-4k} \mod (yk + z)}{yk + z},
\]

\[
B_k \overset{\text{def}}{=} \frac{1}{2^{4k-n-x}(yk + z)}.
\]
Evaluating The Summations

The first sum

\[
\left\{ \sum_{0 \leq k < \frac{n+x}{4}} A_k \right\} = \left\{ \sum_{0 \leq k < \frac{n+x}{4}} \frac{2^{n+x-4k} \text{ mod } (yk + z)}{yk + z} \right\}
\]

- Number of terms: **linear to** \(n\)
- Integer operations: **mod-powering**
- Floating point operations: **division with a fixed precision**
Evaluating The Summations ’

The second sum

\[
\left\{ \sum_{\frac{n+x}{4} \leq k} B_k \right\} = \left\{ \sum_{\frac{n+x}{4} \leq k} \frac{1}{2^{4k-n-x}(yk + z)} \right\}
\]

- Number of terms: linear to the precision
- Integer operations: shifting
- Floating point operations: reciprocal computation with a lower precision
Algorithm Characteristics

For position $n$ and precision $p$,

- **Running time:** $O(p(n^{1+\epsilon} + p))$ for any $\epsilon > 0$
  - $p$ small: essentially linear in $n$, $O(n^{1+\epsilon})$
  - $n$ small: quadratic in $p$, $O(p^2)$
- **Space:** $O(p + \log n)$
- **Embarrassingly parallel:**
  - The summations can be easily split into many smaller summations.
  - Easy to compute in parallel
Parameters

► Usually, we have

- **large position** \( n \) (e.g. \( 2 \cdot 10^{15} \))
- **small precision** \( p \) (e.g. 288)

Again,

- ★ running time is essentially linear, \( O(n^{1+\epsilon}) \);
- ★ space is only \( O(\log n) \).
Errors

▶ Possible errors

- *Rounding errors*: losing precision
- *Hardware errors*: rare but hard to be detected

▶ For the new world record,

- Two computations at two different positions
- Only the bits covered by both computations are considered as valid results.

<table>
<thead>
<tr>
<th>Starting Bit Position</th>
<th>Precision (bits)</th>
<th>Time Used</th>
<th>CPU Time</th>
<th>Date Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>288</td>
<td>23 days</td>
<td>582 years</td>
<td>July 29, 2010</td>
</tr>
<tr>
<td>1,999,999,999,999,993</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,999,999,999,999,997</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>288</td>
<td></td>
<td></td>
<td>503 years</td>
<td>July 25, 2010</td>
</tr>
</tbody>
</table>
Agenda

• Introduction
• A New World Record
• How to Compute The $n^{th}$ Bits of $\pi$?
• Computing $\pi$ with Hadoop
MapReduce Summation

The BBP algorithm basically evaluates the sum

\[ S = \sum_{i \in I} T_i \]

- each term \( T_i \) is simple

\[ \begin{align*}
A_k &= \frac{2^{n+x-4k} \mod (yk + z)}{yk + z} \\
B_k &= \frac{1}{2^{4k-n-x}(yk + z)}
\end{align*} \]

- \( I \) is a large index set

\[ \text{For position } n = 10^{15}, \text{ we have } |I| \approx 7 \cdot 10^{14} \text{ using Bellard’s formula.} \]
A Straightforward Approach

Partition the index set $I$ into $m$ pairwise disjoint subsets $I_1, \cdots, I_m$

Then, compute the summation by a job with

- $m$ maps: each map evaluates

$$\sigma_j \overset{\text{def}}{=} \sum_{i \in I_j} T_i$$

- Single reduce: compute the final sum

$$S = \sum_{1 \leq j \leq m} \sigma_j$$
## Two Problems

- **Multiple maps but one reduce**
  - Fail to utilize reduce slots

- **The job may run for a long time.**
  - Need to persist the intermediate results

### Data Table

<table>
<thead>
<tr>
<th>Starting Bit Position</th>
<th>Precision (bits)</th>
<th>Time Used</th>
<th>CPU Time</th>
<th>Date Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>99,999,999,999,997</td>
<td>1024</td>
<td>4 days</td>
<td>37 years</td>
<td>June 11, 2010</td>
</tr>
<tr>
<td>100,000,000,000,001</td>
<td>1024</td>
<td>5 days</td>
<td>40 years</td>
<td>June 7, 2010</td>
</tr>
<tr>
<td>999,999,999,999,993</td>
<td>288</td>
<td>13 days</td>
<td>248 years</td>
<td>July 2, 2010</td>
</tr>
<tr>
<td>1,000,000,000,000,001</td>
<td>256</td>
<td>25 days</td>
<td>283 years</td>
<td>July 6, 2010</td>
</tr>
<tr>
<td>1,999,999,999,999,993</td>
<td>288</td>
<td>23 days</td>
<td>582 years</td>
<td>July 29, 2010</td>
</tr>
<tr>
<td>1,999,999,999,999,997</td>
<td>288</td>
<td>23 days</td>
<td>503 years</td>
<td>July 25, 2010</td>
</tr>
</tbody>
</table>
Multi-level Partitioning

Partition the sum into many small jobs

Final Sum: \[ S = \sum_{1 \leq j \leq m} \Sigma_j \]

Jobs: \[ \Sigma_j = \sum_{1 \leq k \leq m_j} \sigma_{j,k} \]

Tasks: \[ \sigma_{j,k} = \sum_{1 \leq t \leq m_{j,k}} s_{j,k,t} \]

Threads: \[ s_{j,k,t} = \sum_{i \in I_{j,k,t}} T_i \]

Write the intermediate results into HDFS
Map-side & Reduce-side Computations

► Developed a *generic framework* to execute tasks on either the map-side or the reduce-side.

► Applications only have to define two functions:

- **partition**\((c, m)\): partition the computation \(c\) into \(m\) parts \(c_1, \ldots, c_m\)
- **compute**\((c)\): execute the computation \(c\)
Map-side Job

- Contains **multiple mappers and zero reducers**
  
  - A PartitionInputFormat partitions $c$ is into $m$ parts
  
  - Each part is executed by a mapper
Reduce-side Job

- Contains a mapper and multiple reducers
  - A SingletonInputFormat launches a PartitionMapper
  - An Indexer launches $m$ reducers.
Utilizing The Idle Slices

- Monitor cluster status
  - Submit a map-side (or reduce-side) job if there are sufficient available map (or reduce) slots.

- Small jobs
  - Hold resource only for a short period of time

- Interruptible and resumable
  - can be interrupted at any time by simply killing the running jobs
## Running The Jobs

### Running Jobs

<table>
<thead>
<tr>
<th>Jobid</th>
<th>Priority</th>
<th>User</th>
<th>Name</th>
<th>Map % Complete</th>
<th>Map Total</th>
<th>Maps Completed</th>
<th>Reduce % Complete</th>
<th>Reduce Total</th>
<th>Reduces Completed</th>
<th>Job Scheduling Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_201006091641_93488</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9332</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>97</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 3 running reduce tasks using 3 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93491</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9335</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>96</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 4 running reduce tasks using 4 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93492</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9336</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>92</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 8 running reduce tasks using 8 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93494</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9337</td>
<td>99.99%</td>
<td>200</td>
<td>199</td>
<td>0.00%</td>
<td>0</td>
<td>0</td>
<td>1 running map tasks using 1 map slots. 0 additional slots reserved. 0 running reduce tasks using 0 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93495</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9338</td>
<td>99.99%</td>
<td>200</td>
<td>179</td>
<td>0.00%</td>
<td>0</td>
<td>0</td>
<td>21 running map tasks using 21 map slots. 0 additional slots reserved. 0 running reduce tasks using 0 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93497</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9339</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>36</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 64 running reduce tasks using 64 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93499</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9340</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>30</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 70 running reduce tasks using 70 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93500</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9341</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>52</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 48 running reduce tasks using 48 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93501</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9342</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>23</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 77 running reduce tasks using 77 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93502</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9343</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>45</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 55 running reduce tasks using 55 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93503</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9344</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>28</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 72 running reduce tasks using 72 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93505</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9345</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>99.99%</td>
<td>100</td>
<td>3</td>
<td>0 running map tasks using 0 map slots. 0 additional slots reserved. 97 running reduce tasks using 97 reduce slots. 0 additional slots reserved.</td>
</tr>
<tr>
<td>job_201006091641_93508</td>
<td>NORMAL</td>
<td>tsz</td>
<td>1,999,999,999,999,996-288/P20_3_job9346</td>
<td>99.99%</td>
<td>200</td>
<td>117</td>
<td>0.00%</td>
<td>0</td>
<td>0</td>
<td>83 running map tasks using 83 map slots. 0 additional slots reserved. 0 running reduce tasks using 0 reduce slots. 0 additional slots reserved.</td>
</tr>
</tbody>
</table>
The World Record Computation

▶ 35,000 MapReduce jobs, each job either has:
  • 200 map tasks with one thread each, or
  • 100 reduce tasks with two threads each.

▶ Each thread computes 200,000,000 terms
  • ~45 minutes.

▶ Submit up to 60 concurrent jobs

▶ The entire computation took:
  • 23 days of real time and 503 CPU years
Thank you!