DHTs and Sharding

Aurojit Panda
Announcements
Announcements

- Fill out the Github consent form.
Announcements

- Fill out the Github consent form.
- Fill out the Github account association form.
Announcements

- Fill out the Github consent form.
- Fill out the Github account association form.
- Lab 0 is due tonight. Hopefully all of you are done.
Announcements

• Fill out the Github consent form.
• Fill out the Github account association form.
• Lab 0 is due tonight. Hopefully all of you are done.
• Contacting me: please send e-mail to apanda@cs.nyu.edu
Announcements

- Fill out the Github consent form.
- Fill out the Github account association form.
- Lab 0 is due tonight. Hopefully all of you are done.
- Contacting me: please send e-mail to apanda@cs.nyu.edu
  - I do not check the main NYU e-mail address all that often.
Clarifying Final Project

• Some concern about how projects will be graded.
Clarifying Final Project

• Some concern about how projects will be graded.

• Not really looking to be adversarial: ideal outcome is everyone gets an A.
Clarifying Final Project

- Some concern about how projects will be graded.
- Not really looking to be adversarial: ideal outcome is everyone gets an A.
- Main bar: write a proposal that we both agree on.
Clarifying Final Project

• Some concern about how projects will be graded.

• Not really looking to be adversarial: ideal outcome is everyone gets an A.

• Main bar: write a proposal that we both agree on.
  • Important to scope it so that you have some chance of finishing things.
Clarifying Final Project

• Some concern about how projects will be graded.

• Not really looking to be adversarial: ideal outcome is everyone gets an A.

• Main bar: write a proposal that we both agree on.
  
  • Important to scope it so that you have some chance of finishing things.

  • Important to include something you would enjoy working on.
Clarifying Final Project

• Some concern about how projects will be graded.

• Not really looking to be adversarial: ideal outcome is everyone gets an A.

• Main bar: write a proposal that we both agree on.
  • Important to scope it so that you have some chance of finishing things.
  • Important to include something you would enjoy working on.

• Beyond that main metric is whether you put in the effort.
Clarifying Final Project

• Some concern about how projects will be graded.

• Not really looking to be adversarial: ideal outcome is everyone gets an A.

• Main bar: write a proposal that we both agree on.
  • Important to scope it so that you have some chance of finishing things.
  • Important to include something you would enjoy working on.

• Beyond that main metric is whether you put in the effort.
  • Expect that many projects won't be done in time. That is fine.
Clarifying Final Project

• Some concern about how projects will be graded.

• Not really looking to be adversarial: ideal outcome is everyone gets an A.

• Main bar: write a proposal that we both agree on.
  • Important to scope it so that you have some chance of finishing things.
  • Important to include something you would enjoy working on.

• Beyond that main metric is whether you put in the effort.
  • Expect that many projects won't be done in time. That is fine.
  • Some ideas will fail to bear out. That is also fine.
Some Comments on Summaries

• Thanks for sending summaries, looked over most of them.
Some Comments on Summaries

• Thanks for sending summaries, looked over most of them.

• A few observations about mechanics:
Some Comments on Summaries

• Thanks for sending summaries, looked over most of them.

• A few observations about mechanics:
  
  • Please use the **subject** and **e-mail** address on the website.
Some Comments on Summaries

• Thanks for sending summaries, looked over most of them.

• A few observations about mechanics:

  • Please use the subject and e-mail address on the website.

  • Please write your summary in the body of the e-mail, do not send t.
Some Comments on Summaries

• Thanks for sending summaries, looked over most of them.

• A few observations about mechanics:

  • Please use the subject and e-mail address on the website.

  • Please write your summary in the body of the e-mail, do not send t.

• A few observations about content:
Some Comments on Summaries

• Thanks for sending summaries, looked over most of them.

• A few observations about mechanics:
  • Please use the subject and e-mail address on the website.
  • Please write your summary in the body of the e-mail, do not send t.

• A few observations about content:
  • Sorry about Slicer, it was not as self-contained as I thought it would be.
Some Comments on Summaries

• Thanks for sending summaries, looked over most of them.

• A few observations about mechanics:
  • Please use the subject and e-mail address on the website.
  • Please write your summary in the body of the e-mail, do not send t.

• A few observations about content:
  • Sorry about Slicer, it was not as self-contained as I thought it would be.
  • Would have liked to see a little more on how the papers relate to each other.
Office Hour Etiquette

• Sorry, I know this sounds bad for you, but necessary for my sanity.
Office Hour Etiquette

- Sorry, I know this sounds bad for you, but necessary for my sanity.

- My office hours are on Thursday from 5 - 6PM or by appointment.
Office Hour Etiquette

• Sorry, I know this sounds bad for you, but necessary for my sanity.

• My office hours are on Thursday from 5 - 6PM or by appointment.

• Please don't come to my office for class help outside of these circumstances.
Office Hour Etiquette

• Sorry, I know this sounds bad for you, but necessary for my sanity.

• My office hours are on Thursday from 5 - 6PM or by appointment.

• Please don't come to my office for class help outside of these circumstances.

• Fine to come for other things like:
Office Hour Etiquette

• Sorry, I know this sounds bad for you, but necessary for my sanity.

• My office hours are on Thursday from 5 - 6PM or by appointment.

• Please don't come to my office for class help outside of these circumstances.

• Fine to come for other things like:
  • Research or advice about random things.
Office Hour Etiquette

• Sorry, I know this sounds bad for you, but necessary for my sanity.

• My office hours are on Thursday from 5 - 6PM or by appointment.

• Please don't come to my office for class help outside of these circumstances.

• Fine to come for other things like:
  • Research or advice about random things.
  • Informing me about new/better ice cream places.
Now onto more fun stuff
Two Problems - One Solution?

- Distributed data storage and management.
- Load distribution.
Two Problems - One Solution?

- Distributed data storage and management.

- Load distribution.
DHTs
Some History

• 1999: Shawn Fanning and Sean Parker create Napster
Some History

- 1999: Shawn Fanning and Sean Parker create Napster
- 2000: Metallica + RIAA + other sue Napster.
Some History

• 1999: Shawn Fanning and Sean Parker create Napster

• 2000: Metallica + RIAA + other sue Napster.

• 2001: Napster shuts down network due to injunction.
Some History

• 1999: Shawn Fanning and Sean Parker create Napster

• 2000: Metallica + RIAA + other sue Napster.

• 2001: Napster shuts down network due to injunction.

• What? Why?
Napster: Original Design

- Track 1: Snuffles
- Track 2: Penelope

- Yogi
- Snuffles
- Huckelberry
- Penelope

Track 1
- Track 1

Track 2
- Track 2
Napster: Original Design

- Track 1: Snuffles
- Track 2: Penelope
Napster: Original Design

Yogi

Snuffles

Track 1

Huckelberry

Penelope

Track 2
Centralization affects Survivability

• Easier to shutdown infrastructure.
Centralization affects Survivability

- Easier to shutdown infrastructure.
- Infrastructure is more susceptible to failure.
Centralization affects Survivability

- Easier to shutdown infrastructure.
- Infrastructure is more susceptible to failure.
- But also much easier to build and manage.
An Alternative to Centralization

Yogi

Snuffles

Track 1

Huckelberry

Penelope

Track 2
An Alternative to Centralization

Any problems with this approach?
Distributed Hash Tables

• Rather than replicating data, distribute it across nodes.

• Requirements:
  • Allow lookup without requiring a centralized index.
  • Allow nodes to enter and exit with relative ease.

• Several proposals: Chord, CAN, Kadmelia, ...

• Several in use: Kadmelia used for the BitTorrent trackers.
Chord
Consistent Hashing

- Consider a hash table with $b$ buckets and $n$ items.
Consistent Hashing

• Consider a hash table with $b$ buckets and $n$ items.

• Add a new bucket, so there are now $b + 1$ buckets.
Consistent Hashing

- Consider a hash table with \( b \) buckets and \( n \) items.
- Add a new bucket, so there are now \( b + 1 \) buckets.
- How many items need to move?
Consistent Hashing

• Depends on what hashing algorithm you use.

• But one pretty common strategy (with some additional details missing)
Consistent Hashing

- Depends on what hashing algorithm you use.

- But one pretty common strategy (with some additional details missing)

\[ h[k] = v \]
Consistent Hashing

- Depends on what hashing algorithm you use.
- But one pretty common strategy (with some additional details missing)

```plaintext
h[k] = v
h[hash(k) % len(h)] = v
```
Consistent Hashing

- Depends on what hashing algorithm you use.

- But one pretty common strategy (with some additional details missing)

```
16 buckets

h: _____________________________

h[k] = v

h[hash(k) % len(h)] = v

h[hash(k) % 16] = v
```
Consistent Hashing

• Depends on what hashing algorithm you use.

• But one pretty common strategy (with some additional details missing)

```
h: [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

h[k] = v

h[hash(k) % len(h)] = v

h[hash(k) % 16] = v
```
Consistent Hashing

\[ h[k] = v \]
\[ h[\text{hash}(k) \mod \text{len}(h)] = v \]
\[ h[\text{hash}(k) \mod 16] = v \]
\[ x = h[k] \]
Consistent Hashing

\[ h[k] = v \]
\[ h[\text{hash}(k) \mod \text{len}(h)] = v \]
\[ h[\text{hash}(k) \mod 16] = v \]
\[ x = h[k] \]
\[ x = h[\text{hash}(k) \mod \text{len}(h)] \]
Consistent Hashing

\[ h[k] = v \]
\[ h[hash(k) \% \text{len}(h)] = v \]
\[ h[hash(k) \% 16] = v \]

\[ x = h[k] \]
\[ x = h[hash(k) \% \text{len}(h)] \]
\[ x = h[hash(k) \% 16] \]
Consistent Hashing

\[ x = h[k] \]
\[ x = h[\text{hash}(k) \% \text{len}(h)] \]
\[ x = h[\text{hash}(k) \% 16] \]

**Added 4 buckets**

\[ x = h[k] \]
\[ x = h[\text{hash}(k) \% \text{len}(h)] \]
\[ x = h[\text{hash}(k) \% 20] \]
Consistent Hashing

- $x = h[k]$
- $x = h[\text{hash}(k) \mod \text{len}(h)]$

How many items need to move?

Added 4 buckets

- $x = h[k]$
- $x = h[\text{hash}(k) \mod \text{len}(h)]$
- $x = h[\text{hash}(k) \mod 20]$
Consistent Hashing

Consistent Hashing


- **Monotonicity**: When buckets are added items move from **old** to **new**.
Consistent Hashing


- **Monotonicity**: When buckets are added items move from *old* to *new*.
  - No items move between *old buckets*.
Consistent Hashing


- **Monotonicity**: When buckets are added items move from old to new.
  - No items move between old buckets.

- Also require other properties (e.g., balance) which are more traditional.
Consistent Hashing
Consistent Hashing

• Map each bucket to a point in the unit interval.
Consistent Hashing

- Map each bucket to a point in the unit interval.
  - How? Use a hash function $h_b$. 
Consistent Hashing

- Map each **bucket** to a point in the **unit interval**.
  - How? Use a hash function $h_b$.
- Map each **key** to a point in the **unit interval**.
Consistent Hashing

• Map each **bucket** to a point in the **unit interval**.
  • How? Use a hash function $h_b$.

• Map each **key** to a point in the **unit interval**.
  • How? Use another hash function $h_k$. 
Consistent Hashing

- Map each **bucket** to a point in the **unit interval**.
  - How? Use a hash function $h_b$.

- Map each **key** to a point in the **unit interval**.
  - How? Use another hash function $h_k$.

- Map key $k$ to bucket $b$ such that: $|h_k(k) - h_b(b)|$ is minimal.
Consistent Hashing

• Map each **bucket** to a point in the **unit interval**.
  • How? Use a hash function $h_b$.

• Map each **key** to a point in the **unit interval**.
  • How? Use another hash function $h_k$.

• Map key $k$ to bucket $b$ such that: $|h_k(k) - h_b(b)|$ is minimal.
  • Is this sufficient for monotonicity?
Chord

- Use consistent hashing to build a distributed hash table.
- Keys must be allocated across different nodes.
- Balanced, i.e., no more than \((1 + \varepsilon)K/N\) keys go to a single node.
- Minimize amount of data that needs to be synchronized.
- Especially important when nodes leave or join.
Chord Slow Lookup

![Diagram of Chord Slow Lookup]

- Successor pointer
Chord Slow Lookup

lookup(k)
hash(k) = 7

Successor pointer
Chord Slow Lookup

lookup(k)  
hash(k) = 7

Successor pointer
Chord Join

Successor pointer
Chord Join

Successor pointer

1

2

3: ...
2: ...

35

4: ...
3: ...

8

37: ...
42: ...
0: ...

27: ...

4: ...

→ Successor pointer
Chord Join

1. Predecessor pointer
2. Successor pointer

Nodes:
1. 37: ...
   42: ...
   0: ...
2. 2: ...
3. 3: ...
4. 4: ...
5. 27: ...
6. 8: ...

Connections:
- Node 1 connects to Node 2, Node 3, Node 4, and Node 5.
- Node 2 connects to Node 3, Node 4, and Node 5.
- Node 3 connects to Node 4 and Node 5.
- Node 4 connects to Node 5.
- Node 5 connects to Node 8.
Chord Join

1. 37: ...
2. 42: ...
3. 0: ...
4. ...
5. ...
6. ...
7. ...
8. ...
9. ...
10. ...
11. ...
12. ...
13. ...
14. ...
15. ...
16. ...
17. ...
18. ...
19. ...
20. ...
21. ...
22. ...
23. ...
24. ...
25. ...
26. ...
27. ...
28. ...
29. ...
30. ...
31. ...
32. ...
33. ...
34. ...
35. ...
36. ...
37. ...
38. ...
39. ...
40. ...
41. ...
42. ...
43. ...
44. ...
45. ...
46. ...
47. ...
48. ...
49. ...
50. ...
51. ...
52. ...
53. ...
54. ...
55. ...
56. ...
57. ...
58. ...
59. ...
60. ...
61. ...
62. ...
63. ...
64. ...
65. ...
66. ...
67. ...
68. ...
69. ...
70. ...
71. ...
72. ...
73. ...
74. ...
75. ...
76. ...
77. ...
78. ...
79. ...
80. ...
81. ...
82. ...
83. ...
84. ...
85. ...
86. ...
87. ...
88. ...
89. ...
90. ...
91. ...
92. ...
93. ...
94. ...
95. ...
96. ...
97. ...
98. ...
99. ...
100. ...

Predecessor pointer  ➔ Successor pointer
Chord Leave

1

37: ...
42: ...
0: ...

2

2: ...

3

3: ...
4: ...

35

27: ...

Predecessor pointer  →  Successor pointer
Chord Leave

Predecessor pointer → Successor pointer
Oops: Need successor list to fix this problem
Chord Leave

Predecessor pointer  ➔ Successor pointer
Fingers

3. lookup(k)
hash(k) = 42

m=6 \( (2^m = 64) \)
Fingers

m = 6 \ (2^m = 64)

3. lookup(k)
hash(k) = 42
Fingers

3. lookup(k)
hash(k) = 42

m=6 \( (2^m = 64) \)

\[\text{finger}[k] = \text{succ}(n + 2^{k-1}) \mod 2^m\]
Fingers

\[ \text{finger}[k] = \text{succ}(n + 2^{k-1}) \mod 2^m \]

3.lookup(k)
\[ \text{hash}(k) = 42 \]

m=6 \( (2^m = 64) \)
Fingers

3. lookup(k)
hash(k) = 42

m=6 \(2^m = 64\)

\[\text{finger}[k] = \text{succ}(n + 2^{k-1}) \mod 2^m\]
Fingers

\[ m = 6 \ (2^m = 64) \]

\[ \text{hash}(k) = 42 \]

\[ \text{finger}[k] = \text{succ}(n + 2^{k-1}) \mod 2^m \]
Fingers

\[
3. \text{lookup}(k) = \text{hash}(k) = 42
\]

\[
m=6 \ (2^m = 64)
\]

\[
\text{finger}[k] = \text{succ}(n + 2^{k-1}) \mod 2^m
\]

---

Finger
Fingers

$3.1 \text{lookup}(k)$

$\text{hash}(k) = 42$

$m = 6$ ($2^m = 64$)

$\text{finger}[k] = \text{succ}(n + 2^{k-1}) \mod 2^m$
3. lookup(k)
hash(k) = 42

finger[k] = \text{succ}(n + 2^{k-1}) \mod 2^m

\text{m=6 \ (2^m = 64)}
Fingers

3.lookup(k) = succ(n + 2^{k-1}) \mod 2^m

hash(k) = 42

m=6 (2^m = 64)
Some Concerns

• What happens with simultaneous joins and leaves?

• Can you ever lose track of a node?

• Can you ever get to a state where a node joins and then disappears?
An aside on hashing

- Why SHA-1/SHA-256/...?
An aside on hashing

• Why SHA-1/SHA-256/...?
  • Do we need cryptographic hash functions for this purpose?
An aside on hashing

• Why SHA-1/SHA-256/...?
  • Do we need cryptographic hash functions for this purpose?
  • There are other hash functions that are much faster to compute.
An aside on hashing

• Why SHA-1/SHA-256/...?
  • Do we need cryptographic hash functions for this purpose?
  • There are other hash functions that are much faster to compute.
• Why are cryptographic hash functions well suited to this purpose?
Two Problems - One Solution?

• Distributed data storage and management.

• Load distribution.
Sharding and Load Balancing
Why Shard?

- Services like databases rely on several resources.
Why Shard?

• Services like databases rely on several resources.
  • Storage (to persist), memory (to cache), CPU (to compute), etc.
Why Shard?

• Services like databases rely on several resources.
  • Storage (to persist), memory (to cache), CPU (to compute), etc.
  • An individual node might not have sufficient resources to handle load.
Why Shard?

- Services like databases rely on several resources.
  - Storage (to persist), memory (to cache), CPU (to compute), etc.
- An individual node might not have sufficient resources to handle load.
  - For example, the database might be too big to fit on one server.
Why Shard?

- Services like databases rely on several resources.
  - Storage (to persist), memory (to cache), CPU (to compute), etc.
- An individual node might not have sufficient resources to handle load.
  - For example, the database might be too big to fit on one server.
  - For example, many updates to a single row mean not enough CPU for all.
Why Shard?

• Services like databases rely on several resources.
  • Storage (to persist), memory (to cache), CPU (to compute), etc.

• An individual node might not have sufficient resources to handle load.
  • For example, the database might be too big to fit on one server.
  • For example, many updates to a single row mean not enough CPU for all.

• Sharding is one way to spread the load around.
Load Balancing

- Sometimes one cannot partition the data, and instead must replicate it.
Load Balancing

- Sometimes one cannot partition the data, and instead must replicate it.

- Also provides resilience: no replication no data after failure.
Load Balancing

- Sometimes one cannot partition the data, and instead must replicate it.
- Also provides resilience: no replication no data after failure.
- Load balancing decides who serves any given request.
Load Balancing

- Sometimes one cannot partition the data, and instead must replicate it.
  - Also provides resilience: no replication no data after failure.
- Load balancing decides who serves any given request.
- Who scales the load balancer?
Load Balancing

• Sometimes one cannot partition the data, and instead must replicate it.
  • Also provides resilience: no replication no data after failure.

• Load balancing decides who serves any given request.

• Who scales the load balancer?
  • Interesting question but won't think about this today.
Both Problems are Somewhat Related
Both Problems are Somewhat Related

- Where should a request be routed?
Both Problems are Somewhat Related

• Where should a request be routed?

• Where should something be stored?
Some Useful Properties

- Related data stays together/related requests get routed to the same machine.
Some Useful Properties

• Related data stays together/related requests get routed to the same machine.
Some Useful Properties

- Related data stays together/related requests get routed to the same machine.
Some Useful Properties

- Related data stays together/related requests get routed to the same machine.

```
set x = 2
x = 2
get x
x = 1
```

```
set x = 2
```
Some Useful Properties

- Related data stays together/related requests get routed to the same machine.
Some Useful Properties

- Related data stays together/related requests get routed to the same machine.
Some Useful Properties

- Related data stays together/related requests get routed to the same machine.

When are requests/data related?
Some Useful Properties

- Ability to add more capacity without disruption.
How can we do this?
Sometimes Consistent Hashing is Not Enough

• Consistent hashing works to give the previous two properties.
Sometimes Consistent Hashing is Not Enough

- Consistent hashing works to give the previous two properties.

- Bigger problem is that a single data unit, or request might be more popular.
Sometimes Consistent Hashing is Not Enough

- Consistent hashing works to give the previous two properties.
- Bigger problem is that a single data unit, or request might be more popular.
- Common observance for web workloads.
Sometimes Consistent Hashing is Not Enough

- Consistent hashing works to give the previous two properties.
- Bigger problem is that a single data unit, or request might be more popular.
  - Common observance for web workloads.
- Random hashing is insufficient for this.
Sometimes Consistent Hashing is Not Enough

- Consistent hashing works to give the previous two properties.

- Bigger problem is that a single data unit, or request might be more popular.

  - Common observance for web workloads.

- Random hashing is insufficient for this.

  - Very little control over what ends up loaded.
Sometimes Consistent Hashing is Not Enough

- Consistent hashing works to give the previous two properties.

- Bigger problem is that a single data unit, or request might be more popular.
  - Common observance for web workloads.

- Random hashing is insufficient for this.
  - Very little control over what ends up loaded.
  - Too hard to reverse engineer an appropriate hash function.
Slicer Like Solutions

• Retain some structures from consistent hashing.
Slicer Like Solutions

- Retain some structures from consistent hashing.
- Map keys to ranges in some interval.
Slicer Like Solutions

- Retain some structures from consistent hashing.
- Map keys to ranges in some interval.
- Assign key ranges to individual nodes.
Slicer Like Solutions

- Retain some structures from consistent hashing.
- Map keys to ranges in some interval.
- Assign key ranges to individual nodes.
- Respond to different scenarios by
Slicer Like Solutions

- Retain some structures from consistent hashing.
  - Map keys to ranges in some interval.
  - Assign key ranges to individual nodes.
  - Respond to different scenarios by
    - Merging adjacent ranges.
Slicer Like Solutions

• Retain some structures from consistent hashing.
  • Map keys to ranges in some interval.
  • Assign key ranges to individual nodes.
• Respond to different scenarios by
  • Merging adjacent ranges.
  • Partitioning ranges.
A Few Challenges

• Assigning a key to a partition is no longer stateless.

• Need to record this information somewhere.

• Need to make sure that you can safely update this configuration.

• Despite multiple assigners which might fail or arrive at inopportune moments.

• Despite arbitrary delays.

• One of the things the paper addresses using CAS in Section 4.1.
Conclusion

• Problem: How to spread request/data/... across machines.
  • Rather how to ensure that everyone agrees on what machine to go to.
• Elegant solution: Do it with hashing, no state.
  • Elegant but some limits on what can be achieved.
• General solution: Rely on state, enforce any policy on how things are spread.
  • Now you have two problems: how to distribute and keep state in sync?