MIT CSAIL FastCode Seminar – Bradley Kuszmaul, 11/9/2020

1
00:00:03.810 --> 00:00:13.950
Julian Shun: Welcome to the Fast Code Seminar. Today I'm very happy to have Bradley Kuszmaul as our speaker. Bradley received his PhD from MIT in 1994

2
00:00:14.370 --> 00:00:27.690
Julian Shun: Working in the super tech group, and his research focused on computer systems that have good theoretical and practical guarantees. In 1987 he took a year off from grad school to work …

3
00:00:28.740 --> 00:00:34.530
Julian Shun: on the connection machine supercomputer CM five where he was one of the principal architects.

4
00:00:35.130 --> 00:00:46.290
Julian Shun: And then when he returned to MIT in 1990 He co-developed the MIT Cilk multi threaded programming environment, as well as several prize-winning parallel Chess programs.

5
00:00:47.010 --> 00:00:53.940
Julian Shun: In 1995 he joined the faculty at Yale, where he developed a theory of asymptotically optimal super scale processors.

6
00:00:54.540 --> 00:01:03.600
Julian Shun: And then he joined Akamai Technologies and 1999 as a senior research scientist, where he contributed to the network communications infrastructure.

7
00:01:03.990 --> 00:01:19.290
Julian Shun: And then he led the development of the network usage database. In 2002 he returned to MIT as a research scientist in the super tech group where he worked on scalable transactional memory and practical cash oblivious storage systems.

8
00:01:19.830 --> 00:01:21.570
Bradley Kuszmaul: And then in 2016 he joined

9

00:01:21.600 --> 00:01:30.390
Julian Shun: Oracle where he was one of the architects for the Oracle file storage service which Bradley is going to tell us about today, and then more recently…

10
00:01:31.170 --> 00:01:54.120
Julian Shun: in this year, he joined Google and Bradley's a member of the IEEE and ACM and he's also received various awards for his work, best paper awards and so forth. And today he's going to tell us about the file storage service or FSS in Oracle Cloud infrastructure. So I'll turn it over to you Bradley.

11
00:01:55.620 --> 00:02:04.080
Bradley Kuszmaul: Okay, so, um, so can everyone. My pleasure. I’m showing right. Yeah. And if I go before they were ok so I'm going to

12
00:02:04.620 --> 00:02:15.510
Bradley Kuszmaul: Wear the slogan in our group. Everybody loves file because customers all want to store things and files because it's so much easier for them to understand something like a database.

13
00:02:16.680 --> 00:02:29.550
Bradley Kuszmaul: This this paper they cook in the papers. The conference paper in a journal paper and the co authors are Matteo Frigo and Justin Mazzola Paluska and Sasha Sandler.

14
00:02:30.000 --> 00:02:35.580
Bradley Kuszmaul: But there's a bunch of other people without whom this project could not possibly have worked

15
00:02:36.450 --> 00:02:45.060
Bradley Kuszmaul: You know there were there were 30 people in in the Boston team that developed FSS and there were, you know, perhaps, but

16
00:02:45.870 --> 00:03:01.620
Bradley Kuszmaul: Rate, you know, depending on what time you look at it, but between 500 and and 5000 people in the Oracle Cloud infrastructure team that that we're doing all that, you know, we couldn't we couldn't have done it without the, for example, the people who stood up machines and so forth.

17

00:03:03.210 --> 00:03:13.410
Bradley Kuszmaul: It turns out that Matteo and Justin and I are now at Google. And that's part that's kind of you know that the project I think reached a natural ending point. So that's kind of why we moved. Moved on

18
00:03:14.910 --> 00:03:25.260
Bradley Kuszmaul: And so I'm not speaking as a Oracle employee and just giving this talk as an academic talk, but I-node how everything works and they never said I couldn't tell you so.

19
00:03:28.950 --> 00:03:43.860
Bradley Kuszmaul: So what is file storage service. Well, Oracle operate to cloud, it's, it's not that, you know, in principle, it's not unlike, say, the cloud that Amazon runs or Microsoft is your or the Google Cloud

20
00:03:45.000 --> 00:04:03.780
Bradley Kuszmaul: And in that cloud and for the context of this you can set up a file system. And what happens is there's some website you click on and a file system gets created the just your file system and that file system appears on your virtual network as an IP address with an NFL server.

21
00:04:05.010 --> 00:04:18.030
Bradley Kuszmaul: And so you click on this thing and something pops up and you cut and paste the IP address into your NFL configuration file on your on your clients and you can start creating files.

22
00:04:19.890 --> 00:04:26.130
Bradley Kuszmaul: It turns out that Oracle also rent to the NFL client because you need machines to run your application software.

23
00:04:26.670 --> 00:04:42.270
Bradley Kuszmaul: And you pay for what you use. So you pay for the NFL clients that you rent per hour and you you pay some price for the storage and the file system and it was an it's priced as the per gigabyte month so

24
00:04:43.920 --> 00:04:51.210
Bradley Kuszmaul: When I was there. It was it was on the order of 10 or 20 cents per gigabyte month. I don't remember the exact price. Maybe it was higher. Maybe it was 30 cents.

25
00:04:51.780 --> 00:05:01.470
Bradley Kuszmaul: So if you wrote a gigabyte. You would be charged 30 cents per month. And if you wrote another gigabyte. You would be charged 60 cents and if you deleted a gigabyte. You would be back down to 30 cents a month.

26
00:05:02.670 --> 00:05:07.680
Bradley Kuszmaul: And and this was different from it. For example, if you go and rent a virtual disk drive

27
00:05:08.640 --> 00:05:20.070
Bradley Kuszmaul: And this is true on Amazon has a similar product, product offering. If you go and rent a virtual disk drive. You say, Well, I want a one terabyte this drive and they say, Okay, that'll be so much so many dollars per month.

28
00:05:20.580 --> 00:05:25.920
Bradley Kuszmaul: And it doesn't matter how much useful data, you put on to the disk drive they charge you the same amount

29
00:05:27.030 --> 00:05:37.200
Bradley Kuszmaul: For that terabyte, whether you use it or not. So the, so the price is lower per byte just on on when you buy a virtual disk drives in which is when you file because

30
00:05:37.860 --> 00:05:41.340
Bradley Kuszmaul: You know, if you buy a one terabyte drive and only put one gigabyte on it.

31
00:05:42.060 --> 00:05:49.980
Bradley Kuszmaul: Oracle doesn't you know doesn't go and set up a terabyte they wait until you actually write it in order to allocate that that space to us. So they're taking advantage of

32
00:05:50.580 --> 00:06:00.330
Bradley Kuszmaul: Of what some people call 111 phrase for that is oversubscription where you have more you've sold more stuff than you have.

33

00:06:01.020 --> 00:06:08.730
Bradley Kuszmaul: And it kind of sounds unfair or something but oversubscription was known in telecommunications a lot longer.

34
00:06:09.390 --> 00:06:15.840
Bradley Kuszmaul: The long distance company called a trunk right when they gave you this abstraction that you could call cross country.

35
00:06:16.350 --> 00:06:27.180
Bradley Kuszmaul: That there weren't that many phone lines that actually crossed the Mississippi River so that we take advantage of the fact that not everybody wants to call from Boston to San Francisco at the same time, and they would

36
00:06:27.960 --> 00:06:46.020
Bradley Kuszmaul: You know, they were able to make better use of the of the resources. So the game here is we're going to only charge you. The price per gigabyte so we don't get to. We don't get to take it as much advantage of trumping here because if you've written a gigabyte. We have to store that gigabyte.

37
00:06:48.870 --> 00:06:56.730
Bradley Kuszmaul: You start with you start with zero bytes. When you create the file system and you just grows to as many petabytes or whatever is you want dynamically

38
00:06:57.480 --> 00:07:05.040
Bradley Kuszmaul: So that's, that's what file storage service is some level an oracle Oracle and NFL

39
00:07:05.820 --> 00:07:11.970
Bradley Kuszmaul: Does an NFL server that Amazon has, I think, you know, Google may have something like that. I actually don't know what their product is

40
00:07:12.750 --> 00:07:25.980
Bradley Kuszmaul: But it's not unusual to have some sort of filed service that you will. And the best cases that you want it to be scalable. In this way they can go from zero and be charged essentially nothing too huge and be charged the right amount

41
00:07:28.050 --> 00:07:35.310

Bradley Kuszmaul: So here's a, here's a sort of a architectural overview of how we implemented file storage service on the

42
00:07:36.510 --> 00:07:37.980
Bradley Kuszmaul: Can see my mouse pointer.

43
00:07:39.930 --> 00:07:41.160
Julian Shun: Yo, you can see

44
00:07:41.880 --> 00:07:53.310
Bradley Kuszmaul: Over here on the, on the left for these NFL clients that I mentioned to you and we implement presentation host, which is the host that actually gets connected into your private network.

45
00:07:54.000 --> 00:07:56.520
Bradley Kuszmaul: And it talks and Fs over these links.

46
00:07:57.000 --> 00:08:09.030
Bradley Kuszmaul: And these these machines are all in different potentially in different networks so they can't address each other because they're in different address spaces in the network. So these addresses here. These 10 dot zero to 97

47
00:08:09.540 --> 00:08:16.650
Bradley Kuszmaul: Are specific to the network that the client is in the presentation host may be connected to several of these networks.

48
00:08:18.420 --> 00:08:33.630
Bradley Kuszmaul: And so, this guy's job is just to talk FS to the client and speak our internal protocol to the back end machines and the back end machines. There's a whole bunch of storage hosts. So there's presentation host and storage host and the storage host run

49
00:08:34.650 --> 00:08:48.300
Bradley Kuszmaul: Two processes they run something called dendreon AND SOMETHING CALLED DADDY AND vendor in his daddy is essentially the storage layer and dendreon is the file system logic and and hopefully

50

00:08:49.350 --> 00:08:53.220
Bradley Kuszmaul: You know that that's as much detail as I'll give you on this slide.

51
00:08:56.640 --> 00:09:02.040
Bradley Kuszmaul: So that's, that's the architecture of file storage service to the hardware and software overview

52
00:09:03.150 --> 00:09:06.960
Bradley Kuszmaul: Well, before we dive into exactly how that

53
00:09:08.520 --> 00:09:20.250
Bradley Kuszmaul: how all of that works when could you know we were constantly challenged with, why not do something simpler. Why create this complicated system with, you know,

54
00:09:21.780 --> 00:09:34.200
Bradley Kuszmaul: Presentation house storage hosts and data is spread around, why not just deploy some servers and they run Linux and each and they connect to some disk drives and each, each one has an NFS server on it.

55
00:09:35.100 --> 00:09:42.600
Bradley Kuszmaul: And you know that you know that you can set something like that up pretty quickly. And in contrast FSS took us

56
00:09:43.170 --> 00:10:00.840
Bradley Kuszmaul: You know, basically took Matteo and me and Justin and Sasha six months of hard work to get it to where it was in sort of, you know, beta condition and six months of Matteo is an awful lot. I don't know if you know Matteo. That's an awful lot of development to get

57
00:10:02.190 --> 00:10:05.880
Bradley Kuszmaul: Whereas if you just go and say, well, let me stand up an NFS server, how hard is that

58
00:10:07.440 --> 00:10:10.530
Bradley Kuszmaul: But there's you know system issues that show up when you do that.

59

00:10:10.830 --> 00:10:18.330
Bradley Kuszmaul: You need to be able to migrate file systems when they grow. So you might have some situation here where you have little file systems. So you got to put several

60
00:10:18.630 --> 00:10:23.220
Bradley Kuszmaul: File Systems per, per server. Otherwise, you're not going to be able to use the server efficiently.

61
00:10:23.820 --> 00:10:37.470
Bradley Kuszmaul: And then later the file system gets big. And maybe it doesn't fit on one server. So you have to come up with some scheme for migrating little files that are a piece of one server to big files that that that spanned several servers.

62
00:10:38.580 --> 00:10:43.740
Bradley Kuszmaul: And it doesn't actually seem like it's that easy when you when you start trying to solve those problems.

63
00:10:46.500 --> 00:10:53.970
Bradley Kuszmaul: Among the things you have to solve is you have to solve fail over. So if you've got one, the server that's got four file systems on it and the server crashes.

64
00:10:55.920 --> 00:11:05.220
Bradley Kuszmaul: customers. Customers don't want the file system. They want the file systems to continue to be available. So you need some way to do fail over and replication and then the cloud.

65
00:11:05.850 --> 00:11:17.460
Bradley Kuszmaul: failures happen a lot and I don't know if they happen more in the cloud, and they happen in sort of on premise data centers but they affect a lot more people when they do happen, and

66
00:11:18.750 --> 00:11:25.320
Bradley Kuszmaul: In some ways, clouds are much more reliable because you have a professional team that's dealing with disasters, all the time.

67
00:11:25.830 --> 00:11:35.940

Bradley Kuszmaul: As opposed to if you're if you're some company and you're you've got a data center and nothing bad ever happens then, one day, something bad happens, and nobody nobody has practiced their fire drills, so they don't know what to do.

68
00:11:37.980 --> 00:11:51.870
Bradley Kuszmaul: You have to, you have to pack file systems and migrate them and you have to have a big file systems and I don't really know how to solve that using standard file servers. So that's, that's kind of the problem that we're facing here is that we can't just

69
00:11:52.890 --> 00:11:59.190
Bradley Kuszmaul: At least I don't know how to just stand up file servers and provide them me provide a service in the cloud.

70
00:12:02.580 --> 00:12:12.270
Bradley Kuszmaul: So what did we do well, part of what we did is, when it is we wanted the data structures and the storage system to actually scale.

71
00:12:12.690 --> 00:12:27.480
Bradley Kuszmaul: So we ended up actually building data structures that span across servers and these pick this picture here, the host our storage hosts and these red boxes are some data structure with, you know, the purple lines or pointers in the data structure and

72
00:12:28.680 --> 00:12:37.530
Bradley Kuszmaul: The details of this isn’t a real data structure. This is just sort of drawing of a data structure and we want the pointers to be able to cross servers and

73
00:12:38.010 --> 00:12:52.050
Bradley Kuszmaul: You know, and so it's a global a big global data structure that's kind of the one model one piece of the puzzle for how we how we solve this problem is we get a whole bunch of storage servers and we spread our data structures across those servers.

74
00:12:54.870 --> 00:13:06.330
Bradley Kuszmaul: Well, what data structure do we really want? We decided to just make I-nodes. And so I-nodes were described by Richie and Thompson …

75
00:13:07.560 --> 00:13:13.230
Bradley Kuszmaul: For UNIX, and they are the sort of the abstraction that you have in most Unix like file systems.

76
00:13:14.580 --> 00:13:22.980
Bradley Kuszmaul: An I-node is kind of a little piece of memory that represent the file or directory and inside the I-node there's

77
00:13:23.610 --> 00:13:35.250
Bradley Kuszmaul: There's the metadata like the permissions. So you see RW x here and whether it's a directory. Or here's an I-node that's not a directory. It's just a regular file that's read and writeable.

78
00:13:36.360 --> 00:13:45.870
Bradley Kuszmaul: The I-node has some contents. So in the case of a directory. The I-node it has to have a mapping in here that match from file names to other

79
00:13:47.010 --> 00:13:54.030
Bradley Kuszmaul: I-nodes. So this is the root I-node. And, you know, so we have a path here slash user slash blue dot txt

80
00:13:54.570 --> 00:14:00.270
Bradley Kuszmaul: So there's a U in here and it refers to this I-node. And there were, you know, that would typically be a 90

81
00:14:00.840 --> 00:14:08.010
Bradley Kuszmaul: Number 90 because this is I-node number 90 so that pointer would be represented by the number 90 and then here's another directory, which has a

82
00:14:08.910 --> 00:14:18.570
Bradley Kuszmaul: some contents, which is foo dot text, and then here's a regular file and there's a block map which is which, which has incited a list of blocks and then each of the blocks has some data.

83
00:14:19.920 --> 00:14:25.440
Bradley Kuszmaul: And you can read about that. And there's some clever ideas about how to how to make the block map work.

84
00:14:25.920 --> 00:14:34.230
Bradley Kuszmaul: Direct directories, you have to come up with some way to implement directories and one of the, one of the things that happened about 20 years ago is that is that

85
00:14:35.010 --> 00:14:39.720
Bradley Kuszmaul: Most of the file systems that you use on your laptop moved from a mode where if you had

86
00:14:40.500 --> 00:14:46.920
Bradley Kuszmaul: More than more than 10,000 files in a directory things ground to a halt. Nowadays most file systems can handle.

87
00:14:47.460 --> 00:14:59.490
Bradley Kuszmaul: Huge directories, but 30 years ago, most file systems. Most unit file systems could not because they had algorithms that ran in linear time and to do a search or maybe even quadratic time to do some things

88
00:15:00.990 --> 00:15:09.810
Bradley Kuszmaul: So, but if you ignore the details of how the block map works and how the directory map work. This is the data structure that we wanted to implement

89
00:15:14.730 --> 00:15:25.530
Bradley Kuszmaul: Rather than trying to think of the data structure as a bunch of blocks of memory, we decided that we wanted to represent all this stuff.

90
00:15:26.130 --> 00:15:31.410
Bradley Kuszmaul: All this information in tables that were like it was as though it were a database.

91
00:15:31.890 --> 00:15:40.290
Bradley Kuszmaul: So this data structure here with all these lines and boxes can be represented equivalency with this, these three tables. So there's a table of I-nodes.

92
00:15:40.890 --> 00:15:45.810

Bradley Kuszmaul: Here it says, here's a table design node number zero and it has a permission string and an owner

93
00:15:46.650 --> 00:16:07.350
Bradley Kuszmaul: I-node number 42 I don't 90 and down here is is the directory table and this, this is a mapping that comprises the keys are. I remember his name. So we have zero and you and it says 90 and 92 dot txt and then there's a block table that tells you for a given block offset.

94
00:16:08.460 --> 00:16:14.250
Bradley Kuszmaul: I guess I've got some punctuation area here to block them you know block number zero of I-node 42 would be

95
00:16:15.810 --> 00:16:23.160
Bradley Kuszmaul: And that should be a block number nine, number nine, number nine, right, you give this talk, several times and you discover errors, every time.

96
00:16:24.750 --> 00:16:42.390
Bradley Kuszmaul: So okay, so that that's that's how you that's that's sort of the, the sketch of how you represent I-node as tabular data and the same information is there except that once it's tabular you can do things like put it in the country so we we created a big, big tree and

97
00:16:44.040 --> 00:16:55.980
Bradley Kuszmaul: The tree. It's just a search tree that's organized with many, many children per node you can sort of a binary search tree where you'd only have two children for know to be true might have 100 or 1000 children and a node.

98
00:16:58.410 --> 00:17:06.720
Bradley Kuszmaul: And so we have these key these key value pairs which I've just to notice a b c d e f g h and they're sorted by the key.

99
00:17:08.040 --> 00:17:21.780
Bradley Kuszmaul: And we also maintain a double a link list of I'm believe that kind of a, b tree and all the data is stored in believe so technically this is a b plus tree or maybe it's the beast archery, I, I've never been impressed with the the the sort of the

100
00:17:23.130 --> 00:17:28.110

Bradley Kuszmaul: The, the approach that people came up with for naming a bunch of trees that have the same as fantastic

101
00:17:28.530 --> 00:17:37.020
Bradley Kuszmaul: Performance and giving them all slightly different names when they're just variants on the same thing. So the trees and be star trees and being clustering. They're really all the same and

102
00:17:38.130 --> 00:17:52.560
Bradley Kuszmaul: At some level, doesn't really doesn't really affect the as fantastic. So we put all this stuff in a b tree and these nodes of the b tree then are spread across the machines. So you might have this note on one machine and this note on another machine.

103
00:17:53.970 --> 00:18:02.550
Bradley Kuszmaul: And then these nodes. Could they could all be on different machines or them all might happen to be on the same machine essentially laid out randomly to first order.

104
00:18:06.240 --> 00:18:09.840
Bradley Kuszmaul: We employed just one big being tree. So one of the one of the

105
00:18:10.860 --> 00:18:22.590
Bradley Kuszmaul: Decisions that we we made was, you know, one of the challenges is, it shouldn't we have a b tree for every customer or should we have a shared the tree and the problem that we

106
00:18:23.820 --> 00:18:31.560
Bradley Kuszmaul: Know when we thought it would be nice to have a separate vietri for every customer. So if we if we screw up a beach really only screw up one customer

107
00:18:32.790 --> 00:18:42.450
Bradley Kuszmaul: screwing them all up with. Otherwise, we have the situation where there's this one data structure and we we make a mistake, and we could lose all the file systems in, you know, in a data center.

108
00:18:44.070 --> 00:18:53.370

Bradley Kuszmaul: The problem is that the trees that if we have a whole bunch of trees that I need some sort of data structure to keep the b tree and it doesn't seem like it actually solves very much to have a

109
00:18:54.240 --> 00:18:59.370
Bradley Kuszmaul: A, b tree of the trees, instead of just having a bee tree. So we decided just to go with one country.

110
00:19:00.870 --> 00:19:09.990
Bradley Kuszmaul: One. One advantage of doing that is that little file systems just turn out to be like one or two key value pairs in the country. So it only cost you a couple bites.

111
00:19:10.530 --> 00:19:17.070
Bradley Kuszmaul: To create an empty file system and we just let customers. Create File Systems willy nilly, because it's like doesn't cost us anything.

112
00:19:18.300 --> 00:19:25.920
Bradley Kuszmaul: And even really small file systems end up distributed across many services, you know, an empty cloud system doesn't. But as soon as you have more than a few kilobytes.

113
00:19:27.420 --> 00:19:30.270
Bradley Kuszmaul: Of data, you'll end up using multiple servers so

114
00:19:31.980 --> 00:19:33.720
Bradley Kuszmaul: Things things spread out quite quickly.

115
00:19:34.740 --> 00:19:40.110
Bradley Kuszmaul: And we don't we don't have the problem of trying to manage a bunch of retreats. We just have one victory.

116
00:19:44.730 --> 00:19:46.560
Bradley Kuszmaul: To update the b tree.

117
00:19:48.630 --> 00:20:00.360

Bradley Kuszmaul: Requires doing atomic operations across the server. So here's a, here's a little update where I'm going to split it out. I have this node which has a, b, c, d in it and I decided for one reason or another that I want.

118
00:20:01.140 --> 00:20:07.050
Bradley Kuszmaul: A, B, and C and D to be in different nodes. So I want to change it to to note he having

119
00:20:07.920 --> 00:20:20.910
Bradley Kuszmaul: This root node having three children, instead of two. So I gotta go to the free list and find a block in the free list. I gotta move CMT into that note I got a fiddle around with these blue pointers and this purple planner and I gotta fix this winter.

120
00:20:22.320 --> 00:20:28.800
Bradley Kuszmaul: We gotta do a whole bunch of pointers. I don't want to find myself in a situation where if something crashes in the middle.

121
00:20:29.130 --> 00:20:40.650
Bradley Kuszmaul: That I've got half of half of these updates done that would be very painful to write FFC K in it or something in a distributed environment. We just want this to be an atomic operation.

122
00:20:46.380 --> 00:20:50.670
Bradley Kuszmaul: Okay, let me just make sure everybody's on board still

123
00:20:53.280 --> 00:21:03.660
Bradley Kuszmaul: Okay, I got my chat window up. So if anybody has any questions or anything, feel free to ask this, I'm going to search shift gears a little bit from the data structures to

124
00:21:04.860 --> 00:21:06.270
Bradley Kuszmaul: How we do concurrency now.

125
00:21:07.560 --> 00:21:10.110
Bradley Kuszmaul: So everybody happy with where we are.

126
00:21:12.420 --> 00:21:13.800

Bradley Kuszmaul: Still questions at the moment.

127
00:21:13.890 --> 00:21:14.250
Bradley Kuszmaul: Oh,

128
00:21:14.370 --> 00:21:15.030
Julian Shun: I'll let you know.

129
00:21:15.180 --> 00:21:15.660
Somebody

130
00:21:18.600 --> 00:21:26.310
Bradley Kuszmaul: So could implement these atomic operations. We want to know the atomic change the classic thing that you do.

131
00:21:27.270 --> 00:21:39.090
Bradley Kuszmaul: Is which goes back 40 years now, right, is to or even a little more than 40 years is to use two phase commit. And so the way two phase commit works is no

132
00:21:39.720 --> 00:21:49.020
Bradley Kuszmaul: Little brief tutorial on two phase commit is that you have several servers that are participating in a transaction and you pick one of them to be the coordinator

133
00:21:50.100 --> 00:22:00.540
Bradley Kuszmaul: And so each participant record this little piece of the transaction and marks it as its as prepared, which means it's kind of on the knife edge, it can go forward or it can go backward

134
00:22:01.320 --> 00:22:07.380
Bradley Kuszmaul: Depending on what the coordinator says it wants to do so it's it's it's not committed it is

135
00:22:07.710 --> 00:22:17.160

Bradley Kuszmaul: But it's not a boarded either. It's sort of ready. It's in sort of shorteners cat state where you don't know which way it's going to go to the coordinator says, which way it's going to go.

136
00:22:17.850 --> 00:22:29.070
Bradley Kuszmaul: And and the participants then basic basically places the fate of the transaction into the hands of the coordinator. So when the coordinator here is that everybody has prepared.

137
00:22:29.760 --> 00:22:34.620
Bradley Kuszmaul: Then the coordinator writes down on it stable storage and its database or wherever it has data.

138
00:22:35.370 --> 00:22:41.280
Bradley Kuszmaul: That the transactions committed and once that thing or or could choose to abort it and once that has hit the disk.

139
00:22:42.000 --> 00:22:46.830
Bradley Kuszmaul: Then the transactions committed are awarded, even though the server is the various servers don't know it yet.

140
00:22:47.370 --> 00:22:53.760
Bradley Kuszmaul: They won't find out to later. Later on the servers are told to either roll the transaction forward or backward

141
00:22:54.570 --> 00:23:12.330
Bradley Kuszmaul: And if, if the pretense of this is the notification because there was a network outage or their server crashed or something. It just asks later. So two phase commit is a fairly straightforward idea that for how to get how to do a transaction that involves two or more

142
00:23:14.010 --> 00:23:17.820
Bradley Kuszmaul: Transaction systems that want to make one big transaction system.

143
00:23:20.160 --> 00:23:36.570
Bradley Kuszmaul: So the problem with two phase commit. Is that what if just before you. The, the coordinator has marked the transaction is committed, so everybody's

sitting on the knife edge waiting, do I go forward or back and the server and the coordinator crashes.

144
00:23:38.400 --> 00:23:39.060
And

145
00:23:40.110 --> 00:23:47.370
Bradley Kuszmaul: Maybe it doesn't ever come back. Maybe it's gone forever, or maybe the quick know that can happen. Or maybe you know

146
00:23:48.570 --> 00:23:57.360
Bradley Kuszmaul: Maybe the crash was just after the transaction committed or maybe it was just before maybe the server will come back tomorrow if we just wait a little longer.

147
00:23:57.750 --> 00:24:10.380
Bradley Kuszmaul: The client don't know what to do and they can get stuck forever. So that's the big problem with two phase commit. This is the problem that that land port and separately list Cognos okie

148
00:24:11.640 --> 00:24:12.480
Bradley Kuszmaul: Brian okie

149
00:24:13.530 --> 00:24:15.540
Bradley Kuszmaul: Solved with with

150
00:24:17.160 --> 00:24:28.320
Bradley Kuszmaul: Paxos or view stamp replication, which was at they actually wrote, let's govern and folky actually wrote up you stamp rapid replication, a year earlier in a very similar to Paxos

151
00:24:29.400 --> 00:24:33.930
Bradley Kuszmaul: Lambert didn't really write it up in 1989. But that's when he, when he invented it.

152
00:24:35.550 --> 00:24:46.980

Bradley Kuszmaul: And so Paxos is a distributed consensus protocol that lets you let's a bunch of servers come to a consensus about whether a transaction commits, for example, and

153
00:24:48.270 --> 00:24:52.230
Bradley Kuszmaul: I'm not really going to explain packets in detail. But the point is,

154
00:24:52.830 --> 00:25:01.770
Bradley Kuszmaul: Is that it deals with all these cases of what if a server crashes and comes back or what if a message gets lost in the network or maybe if message gets duplicated in the network.

155
00:25:02.250 --> 00:25:12.990
Bradley Kuszmaul: Or a message gets delayed arbitrarily all these bad things. And these bad things all really do happen, as far as I can tell, especially you know when these when you go up to this kind of scale.

156
00:25:13.380 --> 00:25:25.530
Bradley Kuszmaul: That we built the systems in all these bad things happen all the time even inside one data center, all the bad things happen. And if you're doing transactions across wide area networks, the way Google does you know the way

157
00:25:26.580 --> 00:25:38.070
Bradley Kuszmaul: You know when when you go and edit a Google document, for example, your edit you're doing transactions that span large distances, then even more horrible things happen more often.

158
00:25:39.660 --> 00:25:52.500
Bradley Kuszmaul: So Passos it basically lets you handle contents on a single value like does this transaction commit, and then there's a thing called multi packs of that lets you to come up basically build up a log of

159
00:25:53.880 --> 00:25:58.380
Bradley Kuszmaul: Operations and then once you have a log, you can build a state machine and then

160
00:26:00.930 --> 00:26:05.820

Bradley Kuszmaul: And then you're off to the running. You've got the state machine that's distributed across several machines.

161
00:26:06.330 --> 00:26:18.180
Bradley Kuszmaul: And among the among the details that Paxos deals with is things like, well, what if I want to add a machine to the group of, you know, where I want to move the storage from one for that.

162
00:26:18.780 --> 00:26:22.860
Bradley Kuszmaul: You know, I've got five machines, each holding the state and I want to move

163
00:26:23.820 --> 00:26:35.340
Bradley Kuszmaul: The data from one of the machines to a different machine. So I can shut this machine down because it's starting to fail or or something like that. So Paxos lets you deal with all that stuff in ways that work. Right. And basically,

164
00:26:36.840 --> 00:26:38.730
Bradley Kuszmaul: CAN'T. CAN'T GET can't mess you up.

165
00:26:40.770 --> 00:26:52.170
Bradley Kuszmaul: One question I often get asked is why don't we use raft and this I get asked even inside or because or not so much at Google, but inside Oracle

166
00:26:53.550 --> 00:27:01.260
Bradley Kuszmaul: And raft is basically a very into Paxos and the big claim of raft is that it's easier to understand.

167
00:27:02.490 --> 00:27:10.830
Bradley Kuszmaul: And I'm kind of skeptical of that claim, but the claim was backed up by little user study and the user study involved teaching undergraduate Paxos

168
00:27:11.520 --> 00:27:16.950
Bradley Kuszmaul: For an hour and then teaching a different group of undergraduates wrath for an hour and then testing them to find out.

169
00:27:17.670 --> 00:27:33.990

Bradley Kuszmaul: How well they understood practice and racked and the rest students did better. And so one one in a bug in that methodology as well. The teachers who taught it light rafts though maybe it's not surprising that they that the students ended up learning raft better

170
00:27:35.940 --> 00:27:36.420
Bradley Kuszmaul: But

171
00:27:38.160 --> 00:27:48.270
Bradley Kuszmaul: There is a question in my mind is whether raft is really even any different from Paxos because if you look at the messages that go around and rafters. The same messages that happened in packs of

172
00:27:49.890 --> 00:27:57.060
Bradley Kuszmaul: The so at that level Russian Paxos or just the same thing. It's just a different way of explaining the same algorithm.

173
00:27:59.040 --> 00:28:03.810
Bradley Kuszmaul: But the raft people really want tries hard to make an argument that raft is different.

174
00:28:04.890 --> 00:28:16.650
Bradley Kuszmaul: And to make them maybe saying it's not as too much. So if there is a difference. The difference is that that raft. It uses adding to the log as a primitive

175
00:28:17.730 --> 00:28:25.320
Bradley Kuszmaul: So the primitive thing you do unwrapped it, let's all collectively add to the pen this item to the log

176
00:28:27.090 --> 00:28:37.920
Bradley Kuszmaul: Whereas Paxos the primitive let's all come to a consensus on this value. So Paxos can run and constant space because you can allocate all the buttons you need in advance and just

177
00:28:38.430 --> 00:28:46.770
Bradley Kuszmaul: Run Paxos do so even no matter how many abort tap and Paxos can run in that conference space where it's wrapped

178
00:28:47.130 --> 00:29:03.840
Bradley Kuszmaul: Up of what's happened, what happens is keep things keep getting added to the wall, and you actually can't cannot do wrapped in bounded space. And so I, you know, to the extent that raft IS DIFFERENT PERSPECTIVE IT'S buggy because it can't run in bounded space and

179
00:29:05.910 --> 00:29:15.300
Bradley Kuszmaul: I don't know, maybe the probability that it would run that it would use more than you know 10 megabytes or something you can bound the probability away from

180
00:29:16.530 --> 00:29:27.210
Bradley Kuszmaul: close. So close to zero that it doesn't matter, but it really bugs me to have a situation where you were in the environment in the, in the case where things are going bad, you could then run out of space and then end up stuck.

181
00:29:27.960 --> 00:29:36.090
Bradley Kuszmaul: Because you because you've used up all your log space and and and and graphs would only end up using an unbounded amount of space messages were getting lost.

182
00:29:38.190 --> 00:29:44.730
Bradley Kuszmaul: It when things work. Right. It just, it just finishes very just as quickly as anything else.

183
00:29:45.780 --> 00:29:47.790
Bradley Kuszmaul: So that's my rant about raft.

184
00:29:49.140 --> 00:29:56.700
Bradley Kuszmaul: Is that you should. I don't believe that it's actually harder to understand Paxos than rock you kind of have to when you read the packs of papers, you have to

185
00:29:57.180 --> 00:30:06.810
Bradley Kuszmaul: get over the fact that Lampert wrote the sensible paper about, you know, these ancient Greeks and so forth and see through to the underlying algorithm.

186

00:30:07.560 --> 00:30:18.630
Bradley Kuszmaul: And some, some people apparently really disliked that that approach to writing papers, but I don't find that that that I don't think that the papers were that hard to read. So anyway,

187
00:30:20.730 --> 00:30:27.600
Bradley Kuszmaul: Okay, so we use Paxos because we don't use raft and the only choices are taxes or RAFT WHICH IS packs.

188
00:30:30.510 --> 00:30:46.410
Bradley Kuszmaul: We and we use it to implement two phase commit. So we, the state machine that we built. We Paxos lets you build up a state machine any state machine you can program you can program it and do whatever you want. So the state machine that we made was to implement two phase commit

189
00:30:47.910 --> 00:30:57.840
Bradley Kuszmaul: And just the two phase commit is implemented by this replicated state machine. That means that none of the all the participants and the coordinator and everybody or non stop since the replicated.

190
00:30:59.760 --> 00:31:11.730
Bradley Kuszmaul: You know, we're not, you know, the assumption that we make is that we're not going to have so many a bunch of machines fail and not and failed to come back after you know if there's a power outage or something.

191
00:31:12.870 --> 00:31:16.770
Bradley Kuszmaul: And so we just, you know, so then we do two phase commit and and

192
00:31:18.180 --> 00:31:31.830
Bradley Kuszmaul: And do. So it's sort of relatively complicated Paxos and on top of that is the older simpler thing to think about, which is two phase commit where you, where you have a coordinator for every transaction.

193
00:31:33.360 --> 00:31:33.960
Bradley Kuszmaul: And so

194
00:31:35.190 --> 00:31:40.170

Bradley Kuszmaul: Each each of the replicated participants is called an extent that in our system.

195
00:31:41.400 --> 00:31:52.350
Bradley Kuszmaul: And the extent has a bunch of state. So, how big should the extent be well one of the properties is that the state of the extent must fit on a single server. So we don't want it to be too big.

196
00:31:53.340 --> 00:32:06.030
Bradley Kuszmaul: And we also the overhead of that state machine to that extent has to implement two phase commit. So we don't want you know that's a lot of overhead. It turns out, so we don't want an extent to be too small.

197
00:32:07.530 --> 00:32:17.190
Bradley Kuszmaul: So we signed the extent to that hundreds of them fit onto a server, which means that, then we can do load balancing by moving it around and

198
00:32:18.900 --> 00:32:33.900
Bradley Kuszmaul: We can we can exploit parallels and this is a machine that crashes and we need to rebuild a bunch of extents. We can we can we can move, you know, we've moved 100 EXTENT THAT WE'RE ON THE DEAD machine to 100 different machines and use parallelism to speed up recovery.

199
00:32:35.670 --> 00:32:37.920
Bradley Kuszmaul: Our sensor or five way replicated.

200
00:32:39.360 --> 00:32:47.820
Bradley Kuszmaul: And why would I do five way replication. A lot of storage systems out there seem to think three way replication is enough.

201
00:32:48.900 --> 00:32:53.370
Bradley Kuszmaul: Because basically, you know, machine goes down and you can keep going and then

202
00:32:54.000 --> 00:33:01.410
Bradley Kuszmaul: Maybe you need to put up another machine to replace the damn machine because it crashed, or maybe the other machine comes up, so why should I use five way replication.

203
00:33:02.190 --> 00:33:09.150
Bradley Kuszmaul: And the answer is that we know we have to take machines down for maintenance or software upgrades. So there's scheduled times when

204
00:33:09.630 --> 00:33:17.730
Bradley Kuszmaul: When one machine out of the out of the group is down and during that time, we could have a crash. So we want to be able to tolerate one

205
00:33:18.510 --> 00:33:35.490
Bradley Kuszmaul: One unscheduled failure. While we're doing maintenance on another machine. And so we can schedule the five way replication and we do maintenance on one of the, one of the ways at a time. And basically, things can keep, keep going. As long as three machines are up.

206
00:33:36.960 --> 00:33:39.420
Bradley Kuszmaul: Amazon has I think they do.

207
00:33:42.090 --> 00:33:51.720
Bradley Kuszmaul: They do 15 way replication. I think because they actually do the five way replication within one data center and then they have three data centers replicating the same thing.

208
00:33:53.190 --> 00:34:04.470
Bradley Kuszmaul: And we chose we chose not to do cross data center replication at Oracle. And I think that's because we the feedback we got from customers is, they were more concerned about latency.

209
00:34:05.010 --> 00:34:14.490
Bradley Kuszmaul: Then if then if the entire data center burns down, and doesn't come back because they couldn't cope with that. Anyway, the data center burned down having the file system survive.

210
00:34:15.540 --> 00:34:24.510
Bradley Kuszmaul: doesn't solve the whole problem. They need to have a recovery strategy. Anyway, so why should they pay for the cost of doing cross data center replication.

211
00:34:26.850 --> 00:34:27.330
Okay.

212
00:34:29.100 --> 00:34:42.510
Bradley Kuszmaul: On top of two phase commit. So we're going sort of going up the stack here we program something called multi page store condition on this is analogous to this is named analogous to the multi

213
00:34:44.220 --> 00:34:45.060
What is, what is the

214
00:34:48.150 --> 00:34:59.100
Bradley Kuszmaul: There's a local store condition low can't remember the name now in computer processors some machine have a have a code links for conditional again.

215
00:34:59.610 --> 00:35:00.930
Bradley Kuszmaul: Mode linked

216
00:35:01.050 --> 00:35:02.520
Charles Leiserson: store conditional

217
00:35:03.060 --> 00:35:12.240
Bradley Kuszmaul: Link store conditional. So this is yes. Thank you. So, this is this is essentially the same thing. Load link store conditional reads a location.

218
00:35:13.740 --> 00:35:23.760
Bradley Kuszmaul: Modifies it modifies the value in a register and then you store it back and the store only succeeds, the sources to succeed only if the modified

219
00:35:24.390 --> 00:35:35.670
Bradley Kuszmaul: Value if that location hasn't been modified by anyone else. So the same story. Here we have multi page story additional so here instead of reading

220
00:35:36.720 --> 00:35:45.330
Bradley Kuszmaul: One word of memory we read up to 15 pages and we read them at the memory. Memory actors are register in this situation, the pages live out on disk.

221
00:35:45.810 --> 00:35:53.940
Bradley Kuszmaul: We read 15 pages or maybe maybe we don't need all 15 but we read up to 15 and for each page, we would get a version tag that

222
00:35:54.450 --> 00:35:57.240
Bradley Kuszmaul: That's guaranteed to change at the page ever gets modified

223
00:35:57.750 --> 00:36:05.700
Bradley Kuszmaul: We compete with new values for all the pages. So we can we can change all the pointers and you know if we need to modify the bean tree. We can change the pointers and so forth.

224
00:36:06.300 --> 00:36:12.120
Bradley Kuszmaul: We present the set of new pages, along with the version tags that we got at the beginning and we say

225
00:36:12.870 --> 00:36:18.450
Bradley Kuszmaul: Write these back and those rights occur either either they all occur, and they happen automatically.

226
00:36:19.410 --> 00:36:32.760
Bradley Kuszmaul: And or none of them happen. And if any of the changes, any of the pages change. We don't write it back. It's possible that the transaction will fail. Even if even if there's no obvious conflicts, we just

227
00:36:34.740 --> 00:36:47.820
Bradley Kuszmaul: Other other things can go wrong. Besides, besides that the that somebody modified page and conflicted with you. But basically the game then is that multi page store conditional operations are completely linear if

228
00:36:49.740 --> 00:36:53.040
Bradley Kuszmaul: You can think of them as happening in some global order.

229
00:36:54.540 --> 00:36:55.860
Bradley Kuszmaul: And so you can reason about them.

230
00:36:57.030 --> 00:37:12.930
Bradley Kuszmaul: And an MPC a multi page store traditional is a kind of a limited transaction, it's, it's not a big transaction, like in a database where you can do a transaction where you know you can modify gigabytes of data in a single transaction on on a typical database.

231
00:37:14.100 --> 00:37:21.270
Bradley Kuszmaul: And it's not too small. It's not like a link store conditional where you, where you get to do a transaction on one word of memory. It's really

232
00:37:21.750 --> 00:37:26.400
Bradley Kuszmaul: Hard to make a bee tree with just two loads and store conditional and people who

233
00:37:26.760 --> 00:37:43.410
Bradley Kuszmaul: Program. This style of this lock freestyle programming often want to have, like, Well, can I do a transaction on two pointers instead of one or something. And we just said, well, we'll give you 15 pages. And so it's big enough to do a b tree, and it's not hard to do the b tree and that's

234
00:37:44.460 --> 00:37:58.260
Bradley Kuszmaul: You know we we let Mattel did the hard work of making multi page or conditional. And then I did it relatively easy work of making the first scalable be treated as far as far as we know, nobody else has made a bee tree that scales out

235
00:37:59.370 --> 00:38:00.390
Bradley Kuszmaul: The way this one does.

236
00:38:06.090 --> 00:38:07.560
Bradley Kuszmaul: skip ahead a little bit

237
00:38:11.400 --> 00:38:15.870
Bradley Kuszmaul: So the, the, there's a simple throughput model here.

238
00:38:18.660 --> 00:38:26.040
Bradley Kuszmaul: Which is that basically every time you add a server that has some disk drives in it. It has some network capacity.

239
00:38:26.670 --> 00:38:38.430
Bradley Kuszmaul: And so you add up all that capacity. And then if you if you then you look, you can look at the requirements that that the workload has on it and we just wanted to see sort of

240
00:38:40.020 --> 00:38:48.210
Bradley Kuszmaul: The, the best case would be if well if you got linear speedup. If I double the number of servers. I can get double the performance. That's kind of what you want.

241
00:38:50.160 --> 00:39:02.220
Bradley Kuszmaul: And it turns out that that model actually works pretty well. So here's a, here's a curve that shows on a little network. This is a, this only has 41 storage servers. So it's like probably

242
00:39:02.790 --> 00:39:09.720
Bradley Kuszmaul: Probably our development network or something. If not, I don't think any of the customer facing networks have so few storage servers in them.

243
00:39:10.500 --> 00:39:19.440
Bradley Kuszmaul: Maybe some maybe some some data center in Taiwan or something or, or, or, or, you know, when it, when it first gets built has this few servers.

244
00:39:20.160 --> 00:39:31.290
Bradley Kuszmaul: The x axis is essentially we increase the number of clients that are pushing workload, the number of NFL clients. So as we go. We start with zero, we get no performance and the y axis is how many

245
00:39:31.620 --> 00:39:40.980
Bradley Kuszmaul: How much bandwidth. We got writing into a file writing into files and as we increase the number of servers. Eventually the performance flattens out and you get

246
00:39:41.820 --> 00:39:51.000
Bradley Kuszmaul: Less less and less performance and the theory would be that the flat part of this curve is is when you're basically maxing out the network, then if the network running at 100%

247
00:39:51.540 --> 00:40:05.250
Bradley Kuszmaul: You, you, you, you wouldn't be able to exceed this number, the actual network numbers are a little higher than this is this is the curve that rather than that the performance of the network and this this part of the curve has to do with what happens

248
00:40:06.360 --> 00:40:23.100
Bradley Kuszmaul: As you know, in the in the domain where you would get linear speedup. And so it looks just like the curve that you would hope for it to look like if you were trying to do linear speedup. So this should be familiar to people who did like the Cilk research like me.

249
00:40:25.380 --> 00:40:26.040
Bradley Kuszmaul: And Charlie.

250
00:40:27.720 --> 00:40:33.480
Bradley Kuszmaul: I'm going to end with a couple of quick little a handful of principles, which is kind of wide skipping the other stuff.

251
00:40:35.850 --> 00:40:42.510
Bradley Kuszmaul: One principle is that that we used when we built the system as we wanted to avoid runtime memory allocation.

252
00:40:43.680 --> 00:40:48.180
Bradley Kuszmaul: And long running processes Malloc becomes a problem space gets fragmented.

253
00:40:48.990 --> 00:41:03.600
Bradley Kuszmaul: You can end up running out of memory, even though you think you're only using the amount of memory are allowed because of fragmentation, you run out. So you end up having to back off somehow and use less memory and Malloc itself becomes a performance bottleneck.

254
00:41:05.790 --> 00:41:11.670
Bradley Kuszmaul: It turns out that at Google somebody told me that Google is now like we're an application at Google.

255
00:41:12.750 --> 00:41:20.400
Bradley Kuszmaul: It, it uses something like 30 to 40% of all the CPU cycles that we use at Google just running Malloc.

256
00:41:21.390 --> 00:41:32.400
Bradley Kuszmaul: Malloc free so it's expensive to do Malloc and FSS is organized so that we essentially don't do Malloc we recreate all our buffers and everything at the beginning.

257
00:41:32.790 --> 00:41:42.090
Bradley Kuszmaul: And we don't do another Malloc ever until you know once the service starts. So you don't need a clever Malloc implementation, we just use the system Malloc.

258
00:41:42.660 --> 00:41:56.940
Bradley Kuszmaul: And you can't run out of memory. If you don't, mouth, that's the that's sort of the, the design that the conclusion of this principle. So don't do runtime memory allocation, especially in long running processes.

259
00:41:59.430 --> 00:42:09.810
Bradley Kuszmaul: Another principle is a board early and often. So I mentioned how transactions could abort because there was conflict, they can have worked for lots of other reasons. And in fact, we put in code that makes

260
00:42:10.260 --> 00:42:19.560
Bradley Kuszmaul: One out of 10,000 transactions, just for just for fun. And so it turns out that the error has run all the time in our code in this code.

261
00:42:20.280 --> 00:42:31.200
Bradley Kuszmaul: A lot of software stacks lot of storage stacks have a property that if something bad happens down deep in the storage system that the error isn't handled properly because nobody ever handled that case.

262
00:42:31.860 --> 00:42:38.070
Bradley Kuszmaul: We, we probably don't have very much of that going on and FSS because we run the Eric has all the time.

263

00:42:40.050 --> 00:42:47.250
Bradley Kuszmaul: Saying abort for arbitrary reasons is just incredibly powerful in terms of making the system really robust

264
00:42:49.440 --> 00:43:01.350
Bradley Kuszmaul: We decided to make all transactions hit the desk so that the NFL presentation hosts don't have to have state. It's really hard to make NFL work, work. Well, if

265
00:43:02.670 --> 00:43:06.810
Bradley Kuszmaul: And if you're if you're if you catch some things and haven't written them to disk.

266
00:43:07.200 --> 00:43:17.790
Bradley Kuszmaul: Turns out the NFL clients don't know don't use most of the hooks that the NFL protocol gives you for doing that. So there's not much much much value and make and doing the hard work.

267
00:43:18.210 --> 00:43:26.340
Bradley Kuszmaul: I think there's a research opportunity here, which is how to make a file protocol that tolerates latency NFL. It's not very good tolerating latency.

268
00:43:30.900 --> 00:43:40.860
Bradley Kuszmaul: Okay, another principle right to a model. So if you're if you to LA is this language that Lampert came up with for validating concurrent protocols.

269
00:43:41.310 --> 00:43:51.120
Bradley Kuszmaul: And we were up to LA models for Paxos and are two phase commit and for the snapshot code. And I don't know what else but it whenever you come up with a concurrent algorithm.

270
00:43:52.020 --> 00:44:03.930
Bradley Kuszmaul: Or protocol, it's really helpful to write a model and let it and what it does is it just tries all the possible enter leaving. See what to see if you're you're in variants that you want to hold continue to hold

271
00:44:04.500 --> 00:44:12.600

Bradley Kuszmaul: You get a result this as well. When you you let this thing run for an hour. And now we know that for any sequence of 12 steps in this protocol.

272
00:44:12.930 --> 00:44:20.100
Bradley Kuszmaul: That it works. Right. You don't know that it works for infinite sequences that you can find out for some bounded amount. So you write to LA for everything.

273
00:44:23.610 --> 00:44:27.840
Bradley Kuszmaul: One of the lessons is you got to figure out how to make big transactions smaller

274
00:44:29.580 --> 00:44:38.550
Bradley Kuszmaul: A classic example of this in file system is when you do a directory rename you're not allowed to move a directory into one of its own descendants

275
00:44:39.600 --> 00:44:43.050
Bradley Kuszmaul: And so you have to look. Yeah, you have to somehow

276
00:44:44.160 --> 00:44:56.760
Bradley Kuszmaul: Guarantee this global property they can never that you never made the directory structure into a cycle. And so how do you do that when you can only look at a constant number of things in a transaction. The paper says so you can go find out.

277
00:45:00.030 --> 00:45:11.850
Bradley Kuszmaul: Another lesson if you're doing something in the cloud. You have to solve the multi tenancy problem very early one of the hardest things to get right is supporting lots of customers on the same server.

278
00:45:12.930 --> 00:45:20.310
Bradley Kuszmaul: They the the isolating customers from each other is hard because there's all sorts of ways. It's not just

279
00:45:20.880 --> 00:45:34.710
Bradley Kuszmaul: A security issue, but it's also a performance issue that you end up having situations where one customer can cause another customer to feel feel like the system has gotten slow and customers hate that. And so you really need to

280
00:45:36.150 --> 00:45:47.700
Bradley Kuszmaul: Solve the multi tenancy problem early. I'm not sure that we actually did that great a job on FSS at solving multi tenancy. But I think it's okay but it's like

281
00:45:48.990 --> 00:46:02.430
Bradley Kuszmaul: I think that there's actually some some research opportunity there on how do you do back off. For example, in a way that guarantees that each of the participants gets sort of their fair share, by some some definition of fairness.

282
00:46:04.950 --> 00:46:14.820
Bradley Kuszmaul: Another lesson is separate safety from lightness and performance safety material. It's like safety and whiteness should be separated and I throw in performance.

283
00:46:15.480 --> 00:46:19.650
Bradley Kuszmaul: Safety is the property that every state that the system can reach is correct.

284
00:46:20.580 --> 00:46:27.450
Bradley Kuszmaul: So this is a correctness property lightness says that you can actually get to the state that you want to get to. You can make forward progress.

285
00:46:28.200 --> 00:46:45.390
Bradley Kuszmaul: And often, this is about performance. So an example and file storage service is that we don't actually follow the b tree to find the belief pages we just sometimes we can just guess what the right leaf page was and

286
00:46:46.800 --> 00:46:54.150
Bradley Kuszmaul: And if we succeed in getting that we don't that then we get a performance advantage but somehow we have to validate that in fact the page with the right page.

287
00:46:54.450 --> 00:47:02.550
Bradley Kuszmaul: As part of the transaction and we had to do that validation anyway because of concurrency. Other people might be not modifying the page under underneath us so

288

00:47:02.910 --> 00:47:12.630
Bradley Kuszmaul: We have to validate the page. So we can we can do a performance optimization, which is to not actually follow the link list from the route to the leaves of the tree.

289
00:47:13.920 --> 00:47:15.510
Bradley Kuszmaul: And and

290
00:47:16.560 --> 00:47:21.930
Bradley Kuszmaul: If you know if we guess wrong the transaction will fail and to go back and figure out what we were things went wrong.

291
00:47:22.770 --> 00:47:36.960
Bradley Kuszmaul: In my experience, many people get confused and try to improve performance or correct or or lightness and by changing the safety properties or they, they, you know,

292
00:47:37.500 --> 00:47:51.900
Bradley Kuszmaul: You really need to think about safety. Safety independently. You say, Okay, here's the messages. Here's all the orders, they could happen. And now you might find optimizations, but it's very, very confusing design, if you if you sort of mix them together.

293
00:47:57.390 --> 00:48:10.050
Bradley Kuszmaul: I guess this is probably my last slide, you know the game that we play is to make powerful abstractions. So we spent the effort to get a good page storage system that's replicated by Paxos

294
00:48:11.520 --> 00:48:18.570
Bradley Kuszmaul: can tolerate all the failures that Paxos can tolerate we spent the effort to build a a distributed be tree.

295
00:48:20.520 --> 00:48:31.770
Bradley Kuszmaul: And then, then we spent, you know, then we spent the effort to design, how you lay out files and as tabular data and the game is kind of generally you try to

296
00:48:32.460 --> 00:48:42.240

Bradley Kuszmaul: Instead of finding some simple little abstraction that almost solve the problem we we just wanted to find the powerful obstruction that really solve the general problem and implement it once.

297
00:48:42.810 --> 00:48:51.270
Bradley Kuszmaul: And then build a system that we can read about. So that's the that's sort of our philosophy material is adamant about this.

298
00:48:52.740 --> 00:49:02.250
Bradley Kuszmaul: I sometimes I'm willing to go and do something that's not so elegant. It's what he wants to do but but generally speaking, it

299
00:49:03.270 --> 00:49:07.500
Bradley Kuszmaul: It really pays to find the right instructions that are really powerful obstructions.

300
00:49:09.360 --> 00:49:13.290
Bradley Kuszmaul: And that is my attack.

301
00:49:14.580 --> 00:49:15.960
Julian Shun: Hey, thanks a lot.

302
00:49:18.960 --> 00:49:20.040
Bradley Kuszmaul: If anyone has any.

303
00:49:20.250 --> 00:49:21.720
Bradley Kuszmaul: questions, feel free to

304
00:49:21.810 --> 00:49:26.760
Julian Shun: Speak up or type in the chat. You have a couple minutes for questions.

305
00:49:32.910 --> 00:49:33.600
Bradley Kuszmaul: How do you

306
00:49:34.500 --> 00:49:42.270

Richard Barnes: Get your, your assertions or otherwise produce these arbitrary or sporadic failures, you're talking about and how that

307
00:49:43.380 --> 00:49:46.080
Richard Barnes: Impacts I guess the user experience at the system.

308
00:49:50.100 --> 00:49:52.200
Bradley Kuszmaul: I'm sorry, could you repeat the question.

309
00:49:53.850 --> 00:49:58.230
Richard Barnes: Yes that's wondering how you produce the stochastic arbitrary failures, you were talking to

310
00:49:59.760 --> 00:50:00.270
Bradley Kuszmaul: Users

311
00:50:01.440 --> 00:50:13.920
Bradley Kuszmaul: Don't Mattel wrote this function that says, Are you feeling lucky and one in one in 10,000 times that it returns true and if it returns true we say we return abort from the commit function.

312
00:50:16.500 --> 00:50:27.300
Bradley Kuszmaul: And every everywhere where something could go wrong, we just introduced you know we just introduced these aborts that happen and

313
00:50:30.090 --> 00:50:42.000
Bradley Kuszmaul: Since it the user cat. The end user, the higher levels of the stack can't tell that it whether a boarded because they got unlucky or because

314
00:50:43.800 --> 00:50:48.030
Bradley Kuszmaul: There really was a conflict with some other transaction that was trying to run at the same time.

315
00:50:49.380 --> 00:50:51.030
Richard Barnes: I guess one follow up to this.

316

00:50:51.060 --> 00:51:06.660
Richard Barnes: Is if you have a deep enough stack. And there are enough potential places to abort it seems as though the probability of abortion approaches one. And since the board function as you describe it isn't aware that it's supported previously you could get into a kind of abort loop.

317
00:51:07.950 --> 00:51:11.370
Bradley Kuszmaul: Yeah, so this. So first of all, the stack, not that deep

318
00:51:13.590 --> 00:51:18.690
Bradley Kuszmaul: And generally, we kind of we generally we kind of a board only at the leaves.

319
00:51:20.190 --> 00:51:27.060
Bradley Kuszmaul: And so, and we know we sort of know how many, how big the transactions are so it's some function might get called

320
00:51:28.200 --> 00:51:30.990
Bradley Kuszmaul: You know, five times in order to do a rename

321
00:51:32.190 --> 00:51:37.590
Bradley Kuszmaul: So, you know, if we make if we make the probability be one in 10,000 then you know

322
00:51:38.190 --> 00:51:46.200
Bradley Kuszmaul: The rename operation might fail with probability five and 10,000 so that's that's still okay there is an interesting issue about

323
00:51:46.860 --> 00:51:53.940
Bradley Kuszmaul: How do you do retry and sometimes it's very important that you that you not do retry

324
00:51:54.780 --> 00:52:04.080
Bradley Kuszmaul: Except at the very top level of the software stack. So you don't want function a calling function be which called Sea and the sea sees a failure. It tries again and then

325

00:52:04.530 --> 00:52:14.280
Bradley Kuszmaul: If be seen as a failure. It tries again and then if he sees a failure. It tries again because you can get an exponential number of retreats happening and the system falls apart.

326
00:52:14.910 --> 00:52:23.280
Bradley Kuszmaul: Basically, as soon as something goes wrong down at C or D that failure should be returned, all the way back up to the top and he should do.

327
00:52:24.510 --> 00:52:31.800
Bradley Kuszmaul: Basically exponential back off. You don't want to do nested exponential back off that that that's a, that's a disaster.

328
00:52:33.360 --> 00:52:34.140
Richard Barnes: Thank you so much.

329
00:52:37.050 --> 00:52:39.900
Julian Shun: Thanks. Does anyone else have any questions.

330
00:52:43.140 --> 00:52:48.300
Julian Shun: So I actually had a question. You said you try the trees and

331
00:52:48.720 --> 00:52:50.310
Julian Shun: A whole bunch of other variants

332
00:52:50.400 --> 00:52:54.240
Julian Shun: Does that include the V to the upside country. So you and your

333
00:52:54.300 --> 00:52:54.990
Bradley Kuszmaul: Oh well.

334
00:52:55.080 --> 00:52:55.770
Julian Shun: I'm rigorous work.

335
00:52:55.950 --> 00:53:08.310

Bradley Kuszmaul: On that same we tried a bunch of so be star and b plus you know when you know when I talk about be trees. I don't really distinguish between them being the the epsilon trees are some topically different

336
00:53:09.870 --> 00:53:13.020
Bradley Kuszmaul: So we did not implement a BDD epsilon tree here.

337
00:53:16.890 --> 00:53:22.170
Bradley Kuszmaul: For for various, you know, technical reasons, it didn't make sense in this context.

338
00:53:24.810 --> 00:53:35.160
Bradley Kuszmaul: So we didn't. We did not build a right out to my data structure, we thought hard about it, but we did not build one that were somehow. And it turns out that that decision.

339
00:53:36.420 --> 00:53:49.020
Bradley Kuszmaul: Is a little questionable because we you know we were pushing the SSD is pretty hard. By the time we got done with everything and a right optimize data structure can put a lot less pressure on the SSD.

340
00:53:51.120 --> 00:53:51.510
Bradley Kuszmaul: Is

341
00:53:52.710 --> 00:53:57.360
Bradley Kuszmaul: But we couldn't figure out how to how to play the game with something like a BDD tree.

342
00:53:58.650 --> 00:54:00.210
Bradley Kuszmaul: Okay great, thanks.

343
00:54:01.140 --> 00:54:11.040
Julian Shun: I was also wondering if you could say a little bit more about how how you were you in detail implemented this multi page story conditional efficiently.

344
00:54:13.110 --> 00:54:13.590
Well,

345
00:54:16.320 --> 00:54:25.410
Bradley Kuszmaul: So there is quite a bit of detail in the paper. So to the extent that I can't tell you in the negative time I have left is always the paper.

346
00:54:29.190 --> 00:54:30.450
Bradley Kuszmaul: Yeah, so

347
00:54:38.430 --> 00:54:44.580
Bradley Kuszmaul: Yeah, I'm not sure I'm not sure I you know I don't have a good short answer for that. Oh, good.

348
00:54:46.350 --> 00:54:54.450
Bradley Kuszmaul: Material spent a lot of time thinking about it and so that you know i think that that tells you that probably there isn't.

349
00:54:55.650 --> 00:54:58.230
Bradley Kuszmaul: An easy answer. Sounds good.

350
00:54:58.860 --> 00:55:00.870
Julian Shun: What, what, what day versus

351
00:55:01.020 --> 00:55:18.690
Bradley Kuszmaul: What conference. So it was so there's a there was a paper in us next ATC a year and that in in summer of 2019 and then in transactions on storage. There's a longer version of the paper and and that that appeared

352
00:55:19.770 --> 00:55:21.390
Bradley Kuszmaul: Sometime in the beginning of this year.

353
00:55:22.500 --> 00:55:36.420
Bradley Kuszmaul: Okay, it's, it's, you know, I'm you know material, you know, the authors are as I listed. And I don't remember it's Oracle's files storage system or everybody loves file or something like that. I don't remember the name of the paper.

354
00:55:37.560 --> 00:55:37.950

Julian Shun: Okay.

355
00:55:39.060 --> 00:55:39.510
Julian Shun: Or no

356
00:55:39.600 --> 00:55:41.430
Bradley Kuszmaul: It's probably on my web so

357
00:55:41.880 --> 00:55:47.880
Bradley Kuszmaul: If you go to my web page it and MIT, probably, probably can find the link if it isn't, it should be.

358
00:55:49.830 --> 00:55:50.490
Bradley Kuszmaul: So, sounds good.

359
00:55:53.340 --> 00:55:56.460
Julian Shun: Great, any, any other questions for Bradley.

360
00:56:03.120 --> 00:56:14.670
Julian Shun: Seems like there are no questions now. But I'll leaves us zoom room open in case anyone wants to just hang out and chat with Bradley. If you have to go that's

361
00:56:15.330 --> 00:56:17.820
Bradley Kuszmaul: completely fine as well. Yeah.

362
00:56:18.390 --> 00:56:22.590
Julian Shun: Yeah but yeah let's let's thank Bradley again for the for the talk.

363
00:56:24.360 --> 00:56:27.480
Bradley Kuszmaul: If you do leave ensure that other people can talk

364
00:56:29.100 --> 00:56:30.480
Bradley Kuszmaul: Yeah. So Linda is

365
00:56:31.260 --> 00:56:32.610

Julian Shun: She's the host

366
00:56:32.670 --> 00:56:38.370
Julian Shun: So she's gonna stay on. I'll stay on for a couple minutes for the next meeting.

367
00:56:40.680 --> 00:56:44.100
Bradley Kuszmaul: Well, thank you for the invitation. Yeah, thanks.

368
00:56:44.130 --> 00:56:45.060
Julian Shun: Thanks for giving

369
00:56:46.260 --> 00:56:47.970
Bradley Kuszmaul: Me interesting my verse

370
00:56:50.040 --> 00:56:50.730
Bradley Kuszmaul: Yeah.

371
00:56:51.630 --> 00:56:52.920
Bradley Kuszmaul: Hope we can have

372
00:56:53.310 --> 00:56:56.880
Julian Shun: Have you give a talk in person sometime soon.

373
00:57:03.540 --> 00:57:04.620
Charles Leiserson: Hey, Bradley. How you doing,

374
00:57:05.790 --> 00:57:07.350
Bradley Kuszmaul: Oh, pretty good.

375
00:57:08.730 --> 00:57:10.920
Bradley Kuszmaul: It's, it's been a rough year

376
00:57:13.590 --> 00:57:15.120
Bradley Kuszmaul: The week that lasted a year.

377
00:57:18.600 --> 00:57:18.990
Charles Leiserson: Right.