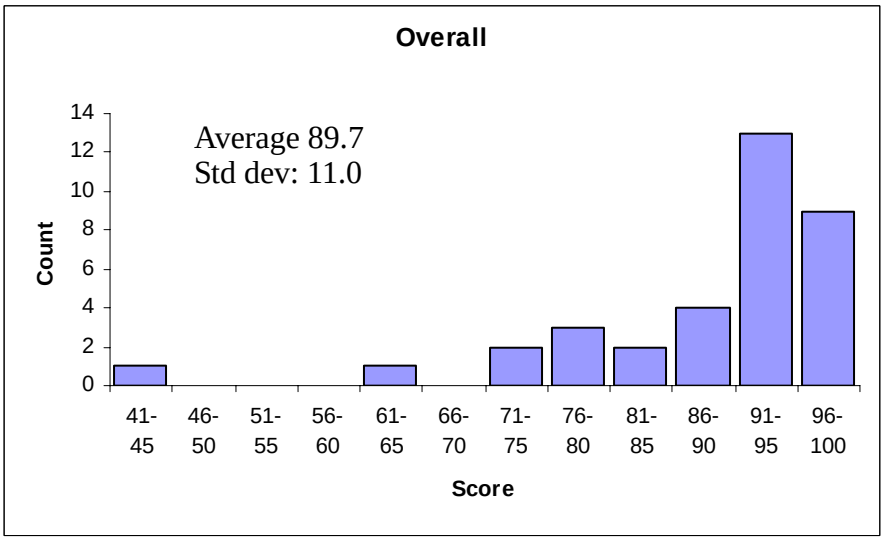
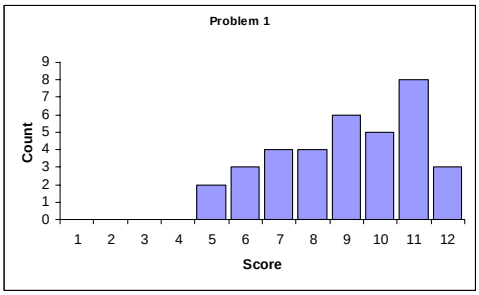
Midterm Solution

This solution set consists of the (roughly) best solutions for each of the problems, taken from actual exams. In many cases, alternative answers were accepted for full or partial credit. My annotations of student solutions are in italics.



1. True/False [12 points]



For each question, answer true or false with a one sentence explanation or example.

a) Bayou's application-specific resolution rules ensure serializability

David Schultz: False. Merge procedures may fail due to unresolvable conflicts among tentative writes. All serial schedules are conflict-free. tence.

b) BFT as described by Castro and Liskov do not work for long-lived systems

Anil Sewani: True: The basic Castro and Liskov BFT algorithm guarantees correct behavior if less than 1/3 of the nodes fail during the lifetime of the system. Long lived systems accumulate faulty nodes over time, which makes the use of their algorithm impractical in such cases (as seen in OceanStore).

c) The geometry of a P2P network matters beyond just flexibility.

Jonathon Hui: TRUE – For example, while most geometries provide the same state-efficiency tradeoff, the butterfly geometry is capable of achieving O(log n) paths with O(1) neighbors. Also, the effect of geometries on the cost of maintaining the overlay structure still remains to be studied.

d) In public-key crypto, it is not necessary to have a private key to sign an item to prove its authenticity.

Bill McCloskey: True. OceanStore uses threshold signatures, where there is no single private key.

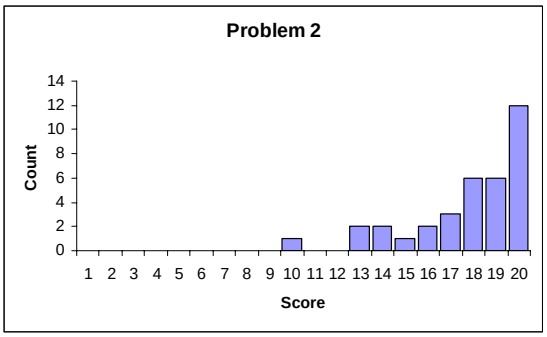
e) Clusters do not suffer from partitions.

Alan Newberger: False. Despite the fact that we can guarantee a lack of partitions in clusters to some probability (e.g. 4 nines), they can in principle suffer from partitions, especially poorly designed ones that do not have fully redundant network switching.

f) Presumed abort helps to avoid livelock in two-phase commit.

Bowie Du: False. Livelock occurs when participating parties cannot make forward progress in the first phase of the 2PC; presumed abort does not help in resolving conflicts before the commit point.

2. Bloom Filters [20 points]



Assume we have a Bloom filter with a bit array of $m = 2^n$ bits, that we use *k* hash functions into the bitmap, and that we are inserting *C* objects. [You can use Excel, Mathematica, etc.]

a) [2] Assuming that we have good hash function (i.e. random bits), then every time we set a bit there is a $p = \frac{1}{m}$ chance that a given bit position will be set. Thus, from the perspective of a specific bit position, this is a *Bernoulli trial*: we flip a biased coin *Ck* times with probability *p* for success (set to 1). What is the probability that a given bit will remain zero at the end?

Manu Sridharan: In a single trial, the probability that a single bit will remain zero is 1 - 1/m. Since our Ck trials are independent, the probability that a bit will remain zero through the trials is therefore (1 - 1/m) ^ (Ck).

b) [3] The probability distribution for a Bernoulli trial is well known; it is the *binomial distribution*, which gives the probability of *x* successes in *N* trials. One way to get *x* successes would be for the first *x* trials to succeed, followed by *N-x* failures, which has probability

$p^x(1-p)^{N-x}$. **Of course, there are** $\binom{N}{x}$ **ways to arrange the *x* successes, which gives us the overall distribution:**

$$Prob(x) = \binom{N}{x} p^x (1-p)^{N-x}$$

Given *C*=100, *n*=10, and *k*=4, what is the probability that a given bit position will have been set more than four times?

Alan Newberger:

The probability that a given bit will have been set more than four times is equivalent to the probability that a bit will not have been set exactly zero, one, two, three or four times. For $C = 100, n = 10, k = 4$, the binomial parameters are $N = Ck = 400, p = 1/2^{10} = 1/1024$. So,

$$\begin{aligned} Prob(>4) &= 1 - Prob(0) - Prob(2) - Prob(3) - Prob(4) \\ &= 1 - 0.677 - 0.263 - 0.052 - 0.0067 - .000649 \\ &= 5.37 \times 10^{-5} \end{aligned}$$

c) [3] Given *C*=100, *n*=10, and *k*=4,

i) What is the expected number of times a given bit will be set to 1?

A.J. Shankar: We compute the expectation to a reasonable number of significant digits: 0 * .6765 + 1 * .2645 + 2 * .0516 + 3 * .0067 + 4 * .0007 = .391 times. (Alternately, we note that there are 1024 bits and 400 1s to go around, so regardless of the distribution, we can simply compute 400/1024 = .391)

ii) What is the probability that no bit has been set more than four times? [three significant digits]

Rodrigo Fonseca: The probability that *a* bit is not set more than 4 times is just 1 - p(x > 4) from above: 1 - 5.37×10^{-5} = 0.9999463. The probability that *no* bit is set more than 4 times is the above event repeated for all 2^{10} bits: 0.9999463^{1024} = 0.947.

d) **[3] As *N* becomes large (and *p* is small), the binomial distribution can be approximated by the Poisson Distribution with parameter** $\alpha = Np$**:**

$$Prob(x) = e^{-\alpha}\frac{\alpha^k}{k!}$$

Using this distribution, estimate the answers to part (c). [As an aside, the way to think about this approximation is analogous to calculus; α **is the mean arrival rate in say, events/hour, and you can slice an hour into *N* intervals each with probability *p* of one event, which is just a Bernoulli trial with *N* flips and thus the expected number of successes is** $Np = \alpha$**.]**

i) *Li Zhuang*: Np = Ck*(1/m) = 0.3906 (same as c)

ii)*Li Zhuang*: The probability of <=4 is:
(Prob(0)+Prob(1)+Prob(2)+Prob(3)+Prob(4))^{1024} = 0.945

e) [3] We would like to be able to delete elements from the Bloom filter. The first way to do this is to use a *q*-bit counter for each position rather than a single bit. Using the Poisson Distribution, what is the probability that we will overflow a 4-bit counter (i.e. *q*=4), with *C*=2000, *n*=10, and *k*=6?

Wei Xu: P1 = Prob(No overflow) = Prob (all bits are "set" 0 to 15 times)
= Sum[E^(-12000/1024)*((12000/1024)^x/x!), {x,0,15}]
= 0.8641

Prob(Overflow) = 1-P1 = 0.136

f) [6, hard] Alternatively, we could use a small value for *q* and use an overflow hash table, H(i), that maps bit positions to integer values. Assume that each entry in H(i) requires 4 bytes. Whenever, we overflow the counter, we leave it at its maximum value and insert a counter into H with value 2^q. We then track the counter value in H until it drops below 2^q, at which point we delete the entry from H and resume using the *q*-bit counter. Given *C*=400, *n*=10, and *k*=6, find the value of *q* that minimizes expected total storage.

Jonathon Hui: The tradeoff in size to consider is the size of the q-bit counter array and the amount of overflow hash table entries that are expected. That is, a smaller q will reduce the size of the q-bit counter array, but also increase the expected number of 4 byte overflow hash table entries. The amount of storage required by the q-bit counter array is simply q * m, where m = 2^n = 1024.

The expected amount of storage required by the hash table is the expected number of counter overflows multiplied by the storage requirements (32 bits) per overflow. To find the probability a given counter overflows, we repeat the calculations using the Poisson Distribution with the given parameters. The expected number of counter overflows is just the probability a given counter overflows times the number of counters m, where m = 2^n. The sum of the q-bit counter array size and the expected amount of storage required by the hash table is the metric we want to minimize.

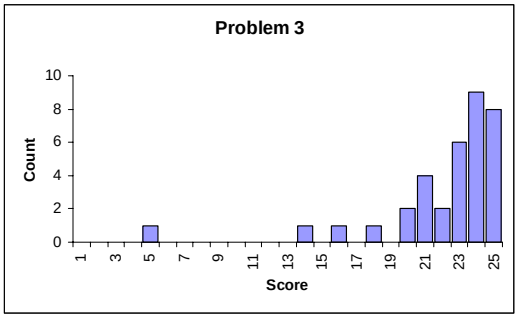
Probability of overflowing a given q-bit counter:
q = 0: 0.9040
q = 1: 0.6791
q = 2: 0.2096
q = 3: 0.0029
q = 4: 4.4047e-009
q = 5: 0

Expected amount of storage from *only* hash entries:
q = 0: 2.9623e+004
q = 1: 2.2253e+004
q = 2: 6.8684e+003
q = 3: 0.0029
q = 4: 4.4047e-009
q = 5: 0

```
Storage used by q-bit array:
      q = 0: 0
      q = 1: 1024
      q = 2: 2048
      q = 3: 3072
      q = 4: 4096
      q = 5: 5120

Expected total storage (in bits):
      q = 0: 2.9623e+004
      q = 1: 2.3277e+004
      q = 2: 8.9164e+003
      q = 3: 3.1670e+003 <-- minimum
      q = 4: 4.0960e+003
      q = 5: 5.1200e+003
Thus, q = 3 minimizes the expected total storage for this configuration.
```

3. Consistency [25 points]



a) [3] In Bayou, what determines the number of rollbacks that a given node must do?

Ojan Vafai: The number of rollbacks is determined by the number of times an earlier write is encountered than exists in the log of the serving node during pairwise anti-entropy, i.e., if the receiver's committed sequence number (CSN) is less than the senders omitted sequence number (OSN).

b) [3] Does application-specific conflict detection/resolution reduce the number of rollbacks?

Ojan Vafai: No, it does not. Since all writes must be serializable in Bayou, the number of roll-backs is dependent on the number of reorderings of writes, not on the number of conflicts. Application specific conflict detection/resolution only reduces the number of conflicts.

c) [4] Make an argument that Bayou is *not* actually highly available.

David Molnar: If tentative writes have ``real actions" as side effects (such as cutting a check or launching a nuclear missile), it may be difficult or impossible to roll back these writes. Therefore, we can't perform these actions until we are sure no rollback will occur. The entire system may need to perform anti-entropy with the primary copy before we can rule out earlier writes, and this may take an arbitrarily long amount of time. Therefore the system cannot be said to be highly available with respect to these real actions.

d) [10] Suppose we want to add real actions to a Bayou application, i.e. actions such as printing a check.

i) [2] How would you change Bayou so that a real action occurs only when you know it won't need to be rolled back?

Byung-Gon Chun: Change Bayou so that a real action occurs only when the write operation tied with the action commits.

ii) [3] Suppose we define an "undo" for printing a check, which is that we signal a human to pull the check from the pile and tear it up. (The "redo" is to reprint the check.) This would allow us to print the check while it is still tentative. Since this is a very expensive undo, explain how we could change Bayou so that we don't repeatedly undo and redo this operation.

Rabin Patra: If we choose to execute tentative updates as well before we know that they are committed, then we may have to rollback and roll forward. However its possible that the same update will be redone again during the roll forward. So to eliminate unnecessary undos, we can maintain the list of updates along with their side-effects. During the rollback, we mark the undo with the update, but also mark the side effects as 'maybe undone'. Then during the roll forward, if we see that the same update is performed again and the side-effect matches, then we don't have to do anything, otherwise we have call for removing the check from the stack.

This is somewhat supported in Bayou with their tuple store, which has 2 bits, the tentative bit and committed bit. During the rollback, the 'print check' operation will have the tentative bit set and the committed bit unset.

iii) [2] Is this system sufficient to ensure that the check is only sent out after it will not need to be undone?

Mike Demmer: No. The optimization in ii) simply avoids unnecessary undo/redo operations in the case where a prior update is actually commutative with the check printing update. To ensure that a check can actually be sent out, i.e. won't be undone ever, requires the solution mentioned in i), wherein attached to the commit operation is a trigger to send the check out of the pile.

iv) [3] Are there situations in which you can tell locally that the "print check" operation will commit before you get the CSN? (so that you know you will never need to undo it)

Rodrigo Fonseca: If you are the primary you know; if you know something about the nature of the operations (for example, if you are the only one emitting checks, while the other clients only check balances and make deposits), or in the more general case when you know the operations are all commutative. Also, if there is a partition in some subset of the data (but in this case you wouldn't need Bayou for that), you could get away with it.

e) [5] Assume that we are using Byzantine fault tolerance to have a group of 3f+1 servers act as the primary copy in Bayou. What would happen in the presence of a partition of the network?

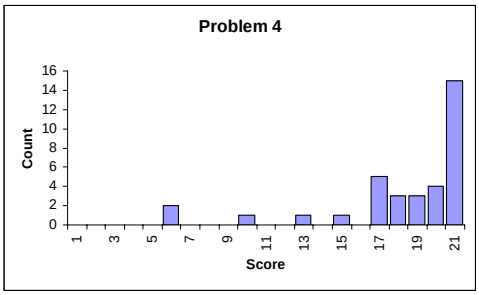
Peter Bodik: I assume we have 3f+1 servers acting as a primary copy (let's call it the "primary cluster"). For the commit to work, we need at least 2f+1 of them to be not faulty and cooperate. If all of the 3f+1 servers get disconnected from some part of the network, the system works fine: machines in the component with the primary copy can work as before and even commit. Machines in the other component(s) can make progress, but the transactions would be commited after the network connects.

If the primary cluster gets split in two (or more) components it's more interesting. If a machine from the primary cluster disconnects, we can regard it as "faulty", since it can't cooperate with the other primary nodes. For the BFT commit to work we need 2f+1 cooperating machines. The disconnected ones can't cooperate and thus are "faulty".

Let's say that the biggest partition of the primary cluster has 2f+1+k nodes, 0<=k<f. In this case, actions might still get commited, but we can't have more than k faulty nodes in that partition.

In partitions with less than 2f+1 we can't commit at all (we have more then f faulty nodes), since in the commit part of the BFT algorithm we need at least 2f+1 nodes. Thus, partitions with less than 2f+1 nodes can't commit and have to wait until the network reconnects.

4. Peer-to-Peer [21 points]



This problem examines a P2P geometry that did not cover much in class: Koorde, http://citeseer.ist.psu.edu/kaashoek03koorde.html

a) [6] Using the degree-2 version of Koorde, explain:

i) What is the PNS value?

Matt Piotrowski: The PNS value is 1 because in order for the invariants of Koorde to hold, a node N must select its two neighbors to be predecessor(2N) and sucessor(N); these nodes are unique, so there is no choice.

ii) What is the PRS value?

Gilman Tolle: Though Koorde does have 2 neighbors per node, the choice of forwarder node is strictly dependent on the key, and thus

```
offers no opportunites for proximity route selection. PRS is also 1.
```

b) [10, hard] Define a m-way multi-Koorde network as a degree-k Koorde network with m links in each of the k directions.

i) **[4] How should these m links be set up?**

Sergui Nedevschi:

i) The k-base de Bruijn graphs connect the node m to nodes km + 1, . . ., km + k - 1. Consequently, every time a branch is taken, a k-base digit is shifted in at the end, and a k-base digit is shifted out from the beginning of the identifier word. We resolve one k-base digits of the identifier at any step, and consequently we need $\log_k(n)$.

In order to find out how can we use m links in each direction, let us analyze the following case: Suppose we we already performed x hops. The identifier word will have $l = \log_k n$ total digits. Thus, at this point, the last x digits are set, while the first l-x digits still need to be resolved.

When taking the next hop, it is clear that we need to maintain the already fixed last x digits, and add the direction digit in the end of the word. However, we do not need to maintain any of the m-x bits at the beginning of the word. Consequently, at this hop, we can link to any words having in the end the x bits + the direction bit:

$|l - x - 1|\ x\ |i|$.

Thus, there are k^{l-x-1} different alternatives. Note, however, that these alternatives are only useful if we are at hops x or less.

We cannot stop to notice a striking similarity with Pastry, where there are a lot of neighbor choices in the first hops, and very few choices at the end.

Since we only have m links, ($< \log_k n$), we propose the following: for each direction i, choose the following links:

- 1st link: - 2k + i, which can be used no matter how many hops have been performed (one choice)
- 2nd link: - an arbitrary first digit, followed by the last l-2 digits in the original word, and then by the direction digit. This next hop can only be used if we are at hop l-1 or less (k neighbor choices).
- 3rd link: - arbitrary first couple of digits, followed by the last l-3 digits, and then by the direction bit. This next hop can only be used if we are at hop l-2 or less (k^2 neighbor choices).

  ..etc.. up to link m.

Note that another, similar approach, would be to make a Can-Koorde hybrid (rather than the Pastry-Koorde hybrid proposed above), where in link m we choose the m-th bit to be discarded. However, this approach is much less flexible.

ii) [4] How would the PNS and PRS values change for this version?

ii) The PNS of the proposed scheme is:
For link 1 in a particular direction, we have only one choice.
For link 2 we have k choices (minus 1 choice already chosen in link 1).
For link 3 we have k^2 choices (minus 2 choices already taken by link 1 and 2).
...
For link m we have k^{m-1} choices ($minus (m-1)$ choices already taken).

Thus, the PNS is the average: $\frac{\sum_{i=1}^{m} k^{m-1}-(m-1)}{m}$

The PRS can be computed similarly:

At the first hop, we have m choices;
At the second hop, we have m choices;
...
At the l-m hop, we have m choices;
At the l-m+1 hop, we have m-1 choices;
...
At the l-th hop, we have 1 choice;

Thus, the PRS is the average: $\frac{\sum_{i=1}^{m} i+(l-m)*m}{m} = \frac{m+1}{2} + (\log_k n - m)$

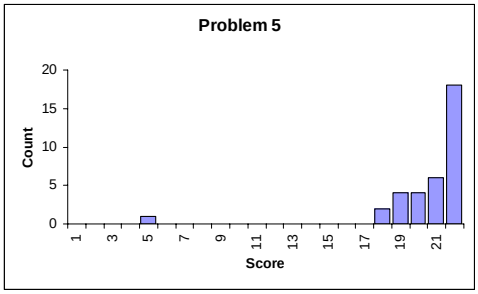
iii) [2] Assuming that m is O(log n), in terms of the performance, which other geometry would be closest to this version?

iii) As we noted before, when $m = log_k n$, the geometry is very similar to Pastry. Consequently, in terms of both PNS and PRS(which dictate performance), the solution is similar to the tree topology.

c) [5] The base-log n version of Koorde is claimed to take O(log n / log log n) hops. Explain why this is.

Shawn Jeffery: For the base-k version of Koorde, each lookup takes $O(\log_k n)$ hops. With k = log n, each lookup takes $\log_{(\log n)} n$ lookups. Using the change of base formula, this can be converted to 1/(log log n). However, with fault-tolerance, each log n step in the routing path takes an extra lookup to find the predecessor of 2m-x. Thus, each lookup takes log n * 1/(log log n), and therefore (with a constant d bits), the total lookup is O(log n/log log n).

5. Federated Systems [22 points]



Assume that we have a working micro-payment mechanism (for real money), and that every DNS lookup requires a small payment to the DNS server.

a) [4] Assuming that quality of service is the same, and that there are many DNS services to choose from, what would be a good mechanism to choose a server (among those mentioned by Mariposa, for example, bidding, purchase orders, batch bids, etc.)?

Feng Zhou: The client should find the lowest price from advertisements on a nameserver and issue a purchase order for the query, or for a bunch of queries if it plans to do more than one.

This is because the DNS query is a simple query and bidding will add too much delay.

b) [8] Assume that the underlying cost depends on the number of hops from the client, and that prices are based on this cost.

i) [1] Assuming that DNS servers do not get overloaded, how would the DNS traffic be spread out?

Shariq Rizvi: Every client will use the DNS server closest to it. DNS traffic will not be a heavy component on major backbones, as most queries will get resolved "locally". Also, a DNS server that has no other DNS servers close to it, will potentially get overloaded.

ii) [1] Assuming that the servers can get overloaded (and servers turn away clients), how would the traffic be spread?

Archana Ganapathi: The nearby servers are more likely to be overloaded than the more distant ones. So client requests to these servers will be dropped more frequently. Requests that cannot afford to wait and retry sending to nearby servers, and can afford to pay more, would have to be directed to servers that are further away. While the cost is higher, chances of request being serviced may be higher. However, the nodes that cannot afford the increased cost will retry nearby servers. So the traffic will still be very high towards nearby nodes (much of it due to lookup retries), but nodes that are further away from clients will receive more traffic (mostly non-retries) than in the overload-less situation.

iii) [2] Argue that this system would make the Internet faster.

Owen Cooper: In i) and ii) we are essentially picking servers based on short network path length, which minimizes the amount of total network bandwidth used for DNS queries. Users will probably also notice faster response times.

iv) [1] When should a DNS operator invest in server capacity?

Rusty Sears: It should run a cost benefit analysis, where the cost is the price of the investment in capacity, and the benefit is the profit that it loses by turning away clients. In other words, unless it is a large provider, it should wait until its systems are incredibly overloaded.

For example, if we assume negligible profit margins, that bidding doesn't take reputation into account, and rejecting clients is free, then adding capacity does not make sense until the system is rejecting so many clients that it will still be unable to service all requests after the new capacity is installed. Therefore, providers will always have at least 1-2 servers less than they need in order to keep up with demand.

v) [3] Assume now that prices reflect both cost and demand, i.e. that overloaded servers charge more to reduce their load but make more profit. Why is this a better mechanism than part (ii).

Bill McCloskey: Some users may not have any other DNS servers that are close by, so they'll be willing to pay the extra cost. Other users may easily be able to switch servers, and the extra cost will cause them to do so. The total effect is that an overloaded server will be used only by those who actually need it, and the load from everyone else will be spread throughout the network. *[Eric: It also causes servers to invest in the right places.]*

c) [4] Why would an enterprise be likely to install their own DNS servers (really caches)?

A.J. Shankar: So that the cost they (via their internal clients) have to pay for DNS resolution goes down dramatically. The private DNS servers just have to query outside servers just once for each domain name (although this process would have to be repeated periodically so that entries don't go stale), and can then serve them up to local clients at very low cost. The benefit is two-pronged: first, the clients don't have to pay an outside agency for lookups, and second, since the local servers are so close, network traffic is reduced, and the cost of sending and receiving the requests is much smaller, since there are fewer hops.

d) [6] DNS uses expiration to invalidate mappings.

i) [2] Why might a client ignore TTLs?

Yatish Patel: A client might ignore TTL because DNS values rarely change, and it would cost money to issue a new lookup.

ii) [4] What would be a reasonable pragmatic policy for expired mappings?

Yatish Patel: Since a majority of DNS entries don't change, a reasonable policy could be to try the expired mapping anyway, and allow the user to force an immediate update if necessary, otherwise periodically refresh the expired mappings at some user specified interval.