Question 1

A. Scale up means making a single node faster, scale out means adding more nodes.

B. Reasons why scale up might not benefit an application:
- might not be possible (e.g. the machine already has the maximum amount of RAM supportable by the architecture)
- price -- might be more expensive than it's worth
- more difficult to share unused resources if load drops

C. Reasons why scale out might not benefit an application:
- makes administration more difficult
- increases probability of failure
- application may not be appropriate for it if it would mean a lot of costly coordination between machines
- higher cost in electricity and cooling

Question 2

Scaling up takes $50 to handle 10 more requests/second.
Scaling out takes $15 to handle 10 more requests/second.
So we would prefer to scale out the system first.

Also notice the system administration cost dominates all other costs. So we prefer to not hire another administrator.

The best approach:
Scale out until xhe limit of 50 machines. Now you can handle 500 requests per second. Cost incurred: 25 * 15 = $375

To handle the rest of the requests, scale up 10 machines for a total cost of 10 * 50 = $500

Total cost: 375 + 500 = $875.

Question 3

A. The system sees the commit point. It executes X = 1100 and Y = 1900.

B. The system doesn't see a commit point for this transaction. So it should do nothing for this transaction.

C. This is not possible. The records in intentions list should be idempotent.

Question 4

We use P(.) to denote probability.
During an operation,

$P(\text{all servers crash}) = 0.05^3 = 0.000125$

$P(\text{exactly two servers crash}) = 3 * 0.05^2 * (1 - 0.05) = 0.007125$

$P(\text{operation fails}) = P(\text{all servers crash}) + P(\text{exactly two servers crash}) = 0.000125 + 0.007125$
$= 0.00725$

Thus, 0.725% of the operations fail.

Question 5

Bug:
The pseudocode updates the salary at a server before sending out success reply to the coordinator. This causes problems if other servers crash during the update process. If any server crashes (or network timed out), no server should be updated, but the pseudocode here will update some of the servers. The simplest solution should recognize the need for local transactions at each server, as well as a single global, 2-phase transaction.

The correct pseudocode should look like this:

```
01. begin local transaction
02. obtain persistent lock on this employee
03. x = current salary of this employee (obtained from the database)
04. x += amount of raise
05. end local transaction (ensure x is written to persistent store)
06. send new salary as success reply to coordinator
07. wait for coordinator's decision
08. if "commit",
      a. begin local transaction
      b. set this employee’s salary in database to x
      c. end local transaction
09. release persistent lock on employee
```

Or like this:

```
01. begin local transaction
02. obtain persistent lock on this employee
03. x = current salary of this employee (obtained from the database)
04. y = x + amount of raise
05. set this employee’s salary in database to y
06. end local transaction (ensure x, database updates written to persistent store)
07. send new salary as success reply to coordinator
08. wait for coordinator's decision
09. if "abort",
      a. begin local transaction
      b. set this employee's salary in database to x
      c. end local transaction
10. release persistent lock on employee
```