1 Distributed Databases: 2PC

Solution Sketch Here, the couple play the role of the clients (subordinates) and the priest plays the role of the co-ordinator.

2PC	**Getting Married**
Request Commit (→)	Do you accept...? (→)
Prepare (AKL) (←)	I do (←)
Commit (→)	I now pronounce you... (→)

2 Yahoo!'s PNUTS

Solution Sketch

DBMS	**PNUTS**
relational model	relational model with simpler query language
indirect addressing (Indexing)	routers + tablets controllers
indexes	routers (similar to a distributed B+-tree)
Write-Ahead Logging	message brokers
bulk loading	bulk loading
data partitioning (vertical or horizontal)	horizontal partitioning
consistency (sync)	consistency (async) with master copy
2PC on records (more frequent)	2PC on tablets (less frequent)

3 map/reduce

Solution Sketch 1. The map function takes a split of the shopping list, and for each item it produces a (store_name, item) pair. The reduce function then merges all the items for a given store, and outputs a list of all the items that can be bought from each particular shop. Then the 14 shops are distributed evenly among the 5 people.

```
map(key shopping_list_name, value shopping_list)
foreach item in shopping_list
   store_name = get_store_name(item)
   emitIntermediate (store_name, item)
end_map

reduce(key store_name, value itemsIterator)
list[item] resultList = new list[item]()
foreach item in itemsIterator
   resultList.append(item)
emit(store_name, resultList)
end_reduce
```

Note that a shopping item may also be available in multiple shops. One could handle that by only returning a single shop in get_store_name(item for each item. Alternatively one could emit entries for all shops. The latter solution allows us to find a better partitioning of the shopping list, however, we sould have to make sure that duplicate items are detected. Otherwise these items will be bought multiple times.

2. In this case, since more than 50% of the items can be bought from one store, the shopping workload will not be balanced among the different people. One alternative to overcome this, is to ensure that for this shop, more than one person shops there. This can be achieved by dividing the stores further into departments. Each person would then take responsability in shopping in departments rather than whole stores. This is done by modifying the map function to produce a composite key (store_name, store_section_name) as follows.

```
map(key shopping_list_name, value shopping_list)
foreach item in shopping_list
    store_name = get_store_name(item)
    store_section_name = get_store_section_name(store_name, item)
    emitIntermediate ( (store_name, store_section_name), item)
end_map
```

The grouping is the done on the composite key. Therefore, reduce() should remain unchanged.

3. To ensure that a balanced shopping load is evenly distributed among the different people, we have to first define a set of cost functions. One such cost function could be assignging costs to individual items,e.g. furniture should have a higher cost than groceries. A second function could assign costs to certain shops, e.g., the cost of going to a far-away store is higher than that of a close store.

Once these cost functions are defined, the groups found in the reduce phase can be assigned to different people. Assigning these groups to people based on the given cost functions can be considered an orthogonal problem to the actual grouping problem. Therefore, the map() and reduce() functions do not have to be changed anymore. Several existing optimization algorithms may be applied.