$$
\begin{aligned}
&+ \quad (\#\ CPU\ storing) * (500\ mgrid\ pgs) * (I/O\ cost) \\
&\quad\ \ 10 * 500 * t_d \\
&\quad\ \ 5,000 * t_d \\
&+ \quad (\#\ CPU\ joining) * (join\ cost) \\
&\quad\ \ 10 * (3 * (500 + 10,000) * t_d) \\
&\quad\ \ 10 * 31,500 * t_d \\
&\quad\ \ 315,000 * t_d \\
&= \quad 5,000 * t_d + 4,500 * t_s + 5,000 * t_d + 315,000 * t_d \\
&= \quad 325,000 * t_s + 4,500 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 500 * t_d + 450 * t_s + 500 * t_d + 31,500 * t_d \\
&= 32,500 * t_d + 450 * t_s
\end{aligned}
$$

8. Print the *eids* of all employees, ordered by increasing salaries. Each processor is connected to a separate printer, and it is acceptable to have the answer in the form of several sorted lists, each printed by a different processor, as long as we can obtain a fully sorted list by concatenating the printed lists (in some order).

   Plan: At each processor, sort the Employees relation by the *sal* field and print the result. Note that the refined Sort-Merge join may be applied without the on-the-fly merge to sort at a cost of $3 * M * t_d$.

$$
\begin{aligned}
Total\ Cost &= (\#\ CPU\ sorting) * (sort\ cost) \\
&= 10 * (3 * 10,000 * t_d) \\
&= 300,000 * t_d
\end{aligned}
$$

$$
Elapsed\ Time = 30,000 * t_d
$$

Exercise 21.4 Consider the same scenario as in Exercise 21.3, except that the relations are originally partitioned using range partitioning on the *sal* and *budget* fields.

Answer 21.4 Range partitioning places tuples with either a *sal* or *budget* field between 0 and 100,000 at processor 1, between 100,001 and 200,000 at the processor 2, etc. The uniform distribution of values in *sal* and *budget* implies that there are equal numbers of tuples at each processor. Then, as is Exercise 21.3, there are 10,000 Departments pages and 500 Employees pages at each processor.

1. Find the highest paid employee.

   Plan: The tuple with the highest *sal* value is located at processor 10. Conduct a complete linear scan of the Employees relation there retaining the tuple with the highest value in the *sal* field.

$$\begin{aligned} Total\ Cost &= (\#\ of\ Emp\ pgs\ at\ CPU\ 10) * (I/O\ cost) \\ &= 10,000 * t_d \end{aligned}$$

$$Elapsed\ Time = 10,000 * t_d$$

2. Find the highest paid employee in the department with *did* 55.

   Plan: Since there is no guarantee that such a tuple might exist at any given processor, conduct a complete linear scan of all Employees tuples at each processor retaining the one with the highest *sal* value and *did* 55. Each processor except one should then send their result to a chosen processor which selects the tuple with the highest value in the *sal* field.

$$\begin{aligned} Total\ Cost &= (\#\ CPU\ scanning) * (\#\ of\ Emp\ pgs/CPU) * (I/O\ cost) \\ &+ (\#CPUs - 1) * (send\ cost) \\ &= 10 * 10,000 * t_d \\ &+ 9 * t_s \\ &= 100,000 * t_d + 9 * t_s \end{aligned}$$

$$Elapsed\ Time = 10,000 * t_d + t_s$$

3. Find the highest paid employee over all departments with *budget* less than 100,000.

   Plan: Department tuples with a *budget* field less than 100,000 must be located at processor 1. The highest paid employees are located at the higher numbered processors, however; as in the part 2 above, there is no guarantee that any processor has an Employees tuple with a particular *did* field value. So, processor 1 must conduct a complete linear scan of Departments retaining only the *did* field. The results are then sent to all other processors which store and join them with the Employees relation retaining only the join tuple with the highest *sal* value. Finally, each processor except one sends the result to a chosen processor who selects the Employees tuple with the highest *sal* value.

$$\begin{aligned} Total\ Cost &= scan\ Dept\ for\ did\ fields\ at\ first\ CPU \\ &+ sending\ did\ field\ tuples\ from\ 1\ CPU\ to\ 9\ CPUs \\ &+ storing\ did\ field\ tuples\ at\ each\ processor \\ &+ joining\ did\ field\ tuples\ with\ Emp \\ &+ sending\ local\ results\ to\ chosen\ processor \\ &= (\#\ CPUs\ w/budget < 100,000) * (\#\ Dept\ pgs) * (I/O\ cost) \\ &\quad 1 * 500 * t_d \\ &\quad 500 * t_d \\ &+ (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (167\ did\ pgs) * t_s \\ &\quad 1 * 9 * 167 * t_s \end{aligned}$$

$$
\begin{aligned}
& 10 * 1,503 * t_s \\
& 15,030 * t_s \\
+ \quad & (\#\ CPU\ storing) * (167\ did\ pgs) * (I/O\ cost) \\
& 10 * 167 * t_d \\
& 1,670 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 10 * (3 * (167 + 1,000) * t_d) \\
& 10 * 3,501 * t_d \\
& 35,010 * t_d \\
+ \quad & (\#\ CPUs - 1) * (send\ cost) \\
& 9 * t_s \\
= \quad & 500 * t_d + 15,030 * t_s + 1,670 * t_d + 35,010 * t_d + 9 * t_s \\
= \quad & 37,180 * t_d + 15,039 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 1,503 * t_s + 167 * t_d + 3,501 * t_d + t_s \\
&= 3,718 * t_d + 1,504 * t_s
\end{aligned}
$$

4. Find the highest paid employee over all departments with *budget* less than 300,000. Plan: The plan is identical to that for part 3 above except that now the first 3 processors must create relations of *did* fields and send them to all other processors.

$$
\begin{aligned}
Total\ Cost \quad = \quad & scan\ Dept\ for\ did\ fields\ at\ first\ three\ CPUs \\
+ \quad & sending\ did\ field\ tuples\ from\ 3\ CPUs\ to\ 9\ CPUs \\
+ \quad & storing\ did\ field\ tuples\ at\ each\ processor \\
+ \quad & joining\ did\ field\ tuples\ with\ Emp \\
+ \quad & sending\ local\ results\ to\ chosen\ processor \\
= \quad & (\#\ CPUs\ w/budget < 300,000) * (\#\ Dept\ pgs) * (I/O\ cost) \\
& 3 * 500 * t_d \\
& 1,500 * t_d \\
+ \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (170\ did\ pgs) * t_s \\
& 3 * 9 * 170 * t_s \\
& 3 * 1,530 * t_s \\
& 4,590 * t_s \\
+ \quad & (\#\ CPU\ storing) * (510\ did\ pgs) * (I/O\ cost) \\
& 10 * 510 * t_d \\
& 5,100 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 10 * (3 * (510 * 1,000) * t_d) \\
& 10 * 4,530 * t_d
\end{aligned}
$$

$$
\begin{aligned}
& & 45{,}030 * t_d \\
& + & (\#\ CPUs - 1) * (send\ cost) \\
& & 9 * t_s \\
& = & 1{,}500 * t_d + 4{,}590 * t_s + 5{,}100 * t_d + 45{,}030 * t_d + 9 * t_s \\
& = & 51{,}630 * t_d + 4{,}599 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time & = & 500 * t_d + 1{,}530 * t_s + 510 * t_d + 4{,}530 * t_d + t_s \\
& = & 5{,}540 * t_d + 1{,}531 * t_s
\end{aligned}
$$

5. Find the average salary over all departments with *budget* less than 300,000.

   Plan: This query is similar to part 4 above. The difference is that instead of selecting the highest salary during the join and reporting to a chosen processor, each processor retains a running sum of the *sal* field and count of joined tuples. The chosen processor then computes the total sum and divides by the total count to obtain the average. Note that the costs are identical to part 4.

6. Find the salaries of all managers.

   Plan: Employees tuples with an *eid* field equal to a *mgrid* field of a Departments relation may be stored anywhere. Each processor should conduct a complete linear scan of its Departments tuples retaining only the *mgrid* field. Then, each processor sends the result to all others who subsequently store the *mgrid* relation. Next, each processor joins the *mgrid* relation with Employees retaining only the *sal* field of joined tuples.

$$
\begin{aligned}
Total\ Cost & = & scan\ Dept\ for\ mgrid\ fields \\
& + & sending\ did\ field\ tuples\ from\ 10\ CPUs\ to\ 9\ CPUs \\
& + & storing\ did\ field\ tuples\ at\ each\ processor \\
& + & joining\ mgrid\ field\ tuples\ with\ Emp \\
& + & sending\ local\ results\ to\ chosen\ processor \\
& = & (\#\ CPUs\ scanning) * (\#\ Dept\ pgs) * (I/O\ cost) \\
& & 10 * 500 * t_d \\
& & 5{,}000 * t_d \\
& + & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (170\ mgrid\ pgs) * t_s \\
& & 10 * 9 * 170 * t_s \\
& & 10 * 1{,}530 * t_s \\
& & 15{,}300 * t_s \\
& + & (\#\ CPU\ storing) * (1{,}700\ did\ pgs) * (I/O\ cost) \\
& & 10 * 1{,}700 * t_d \\
& & 17{,}000 * t_d \\
& + & (\#\ CPU\ joining) * (join\ cost) \\
& & 10 * (3 * (1{,}700 * 1{,}000) * t_d) \\
& & 81{,}000 * t_d
\end{aligned}
$$

$$
\begin{aligned}
& + \quad (\#\ CPUs - 1) * (send\ cost) \\
& \qquad 9 * t_s \\
& = \quad 5,000 * t_d + 15,300 * t_s + 17,000 * t_d + 81,000 * t_d + 9 * t_s \\
& = \quad 103,000 * t_d + 15,309 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 1,530 * t_s + 1,700 * t_d + 8,100 * t_d + t_s \\
&= 9,850 * t_d + 1,531 * t_s
\end{aligned}
$$

7. Find the salaries of all managers who manage a department with a budget less than 300,000 and earn more than 100,000.

   Plan: Department tuples with *budget* less than 300,000 are located at the first three processors. Employees tuples with a *sal* fields greater than 100,000 are located at processors 2 through 10. Conduct a complete linear scan of all Department tuples retaining only the *mgrid* field of tuples with a *budget* field less than 300,000. Send the new *mgrid* relation to processors 2 through 10 which subsequently store them. Next, processors 2 through 10 join the new *mgrid* relation with Employees to obtain the desired result.

$$
\begin{aligned}
Total\ Cost &= scan\ Dept\ for\ mgrid\ fields\ at\ first\ thirty\ CPUs \\
&+ sending\ mgrid\ field\ tuples\ from\ 1\ CPU\ to\ 9\ CPUs \\
&+ sending\ mgrid\ field\ tuples\ from\ 2\ CPUs\ to\ 8\ CPUs \\
&+ storing\ mgrid\ field\ tuples\ at\ 9\ CPUs \\
&+ joining\ mgrid\ field\ tuples\ with\ Emp\ in\ 9\ CPUs \\
&= (\#\ CPUs\ scanning) * (\#\ Dept\ pgs/CPU) * (I/O\ cost) \\
&\quad 3 * 500 * t_d \\
&\quad 1,500 * t_d \\
&+ (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (170\ mgrid\ pgs) * t_s \\
&\quad 1 * 9 * 170 * t_s \\
&\quad 1 * 1,530 * t_s \\
&\quad 1,530 * t_s \\
&+ (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (170\ mgrid\ pgs) * t_s \\
&\quad 2 * 8 * 170 * t_s \\
&\quad 2 * 1,360 * t_s \\
&\quad 2,270 * t_s \\
&+ (\#\ CPU\ storing) * (510\ mgrid\ pgs) * (I/O\ cost) \\
&\quad 9 * 510 * t_d \\
&\quad 4,590 * t_d \\
&+ (\#\ CPU\ joining) * (join\ cost) \\
&\quad 9 * (3 * (510 + 1,000) * t_d) \\
&\quad 9 * 4,530 * t_d \\
&\quad 40,770 * t_d
\end{aligned}
$$

(b) Give all employees a 10 percent raise. After the update is completed, the conditions of the original partitioning of Employees must still be satisfied.

Answer 21.8 1. In the following, when we refer to the "highest paid employee/manager", note there could be duplicates, and the queries will return all such tuples that qualify. Also, we assume there is no cost involved in shipping a query from the site it is posted at, ot the site(s) where it is evaluated.

(a) We need to do a *total exchange* of the Departments partitions. In other words, each site must ship its partition to *each* of the other nine sites. Each of the Departments partition is stored in 500 pages. For each site, the cost of shipping a partition to the other nine sites is $500 \times 9t_s$. Hence, the cost of the *total exchange* is $45,000t_s$. The cost of a local natural join (using Sort-Merge Join) is $3(M + N)t_d$, where $M = 10,000$ Departments records, and $N = 2 \times 10^6$ Employees records. (Remember the Employees relation is also partitioned, so each site has 10,000 pages, each with 200 records.) Hence the cost of the natural join across all the sites is $603 \times 10^5 t_d$. Finally, the result fragments have to shipped to the query site. Each site will have a result fragment of 500 pages (the same size as the local Departments fragment. Why?), and the total cost of shipping these fragments from nine sites to the query site is $4500t_s$. Adding all the costs together, the total cost of the plan is $49,500t_s + 60,300,000t_d$.

(b) Clearly, the highest paid employee will have a salary in the bracket $900,000 < sal \leq 1,000,000$. (We are assured that there is at least one employee in this salary range because the *sal* field has values uniformly distributed in the range 0 to 1,000,000, and there are 100,000 pages of Employees tuples). So we ship the query to the tenth site, evaluate the highest paid employee at that site, and ship the result back to the query site. Finding the employee(s) with the maximum salary at the tenth site requires a scan of the Employees fragment, which costs $10,000t_d$. Since the salary is uniformly distributed, we may assume there are $\frac{200 \times 10,000}{100,000} = 20$ result tuples, which fit on a single page. Thus total cost of the query is $10,000t_d + t_s$.

(c) The best plan for this query results from the observation that the partition $sal \leq 100,000$ is replicated at every site. So we just need to query the partition at the *query site* for the highest paid employee in its partition. This does not involve any shipping costs. Thus, the cost of the query is the cost of scanning the Employees partition, which is $10,000t_d$.

(d) Again, we will find the highest paid employee in the salary bracket $400,000 < sal \leq 500,000$ in the fifth site. So we ship the query to the fifth site, evaluate the query there, and ship the result to the query site. The cost is the same as in part (b) above.

(e) Employees in the salary bracket $450,000 < sal \leq 550,000$ are split across the fifth and the sixth sites. But clearly, by the partitioning of the Employees relation, the employees in the sixth site earn more than those at the fifth site. So we need to query only the partition in the sixth site, but we need to qualify our query by selecting only those employees whose salary is less than 550,000, and then selecting the one with the highest salary. The cost is again the same as in part (b) above.

(f) We need to do a partial join of the two relations. The Departments fragment at the query site is shipped across to each of the other nine sites, and the fragment is joined with the Employees fragment at the site. (Why would SemiJoin or BloomJoin at the query site not buy us much in this case? Hint: The *eid* field is a key for Employees and the join is on this field.) At each site, the result is scanned to select the tuple with the maximum salary, and this tuple is shipped to the query site, where a final scan of the ten tuples contributed by each site yields the department (amongst the ones stored at the query site) with the highest paid manager. The cost of the initial shipping is $4500t_s$, the cost of ten local joins is $603 \times 10^5 t_d$ (refer to part (a)), the cost of the scans is , the cost of shipping the result tuples is $9t_s$, and the cost of the final scan is $10t_d$.

(g) We do the above operation for each fragment of the Departments relation. Thus, the initial shipping costs, and the local join costs increase ten-fold. Everything else remains the same. Thus, the cost of the query is .

2. (a) We use the **read-only write-all** policy for synchronous replication here. Since every site has a copy of the fragment, we can raise the salary of all those with salary less than 100,000 without having to ship any part of the relation. So each site simply raises the salary of all employees in the partition by 10have a salary greater than 100,000, and we just reset their salary to exactly 100,000. In order to do this, each site needs to obtain an exclusive lock on its local copy of the fragment, and the query is executed at each site.

   (b) We again use the **read-only write-all** policy, and we need to obtain an exclusive lock on all the copies of the fragment with employees whose salary is less than 100,000. The remaining fragments of the relation are not replicated, and hence, just one exclusive lock for each fragment is required, at the site where it is stored. Once the update is complete, we need to do some shipping to satisfy the original partitioning of Employees. Each site will ship to the "next" site employees whose salaries have been raised to the next bracket of the original partitioning. In other words, the first site will ship all records will salaries greater than 100,000 to the second site, the second site will ship all records will salaries greater than 200,000 to the third site, and so on. To do this, each site needs to get an exclusive lock on the fragment they store. Once each site has modified its relation, shipped the appropriate records, and has received its set of additional records, it obtains a lock on its fragment to append the new records. The only tricky fragment to handle is the one that has the employees with the lowest salaries, since it is replicated at all the sites. Since we are using synchronous replication, the changes made at the first site must be executed at all the sites. Hence, each site must obtain an exclusive lock on its copy of the fragment, and run the query against it.

3. (a) Using *primary site* asynchronous replication, only the site with the primary copy of the fragment whose employees earn less than 100,000 is visited. None of the other copies need to be locked. The primary site is predetermined, and in our case, it is possibly the first site. The changes to the primary copy will be captured at a latter time, and applied to the secondary copies.

   (b) We agin use *primary site* asynchronous replication. We follow a procedure very similar to the one described when synchronous replication is used. The only difference is in teh handling of the fragment which is replicated at all the sites. Since we are using asynchronous replication here, we can just lock and change the primary copy, and copy the changes to the other sites at some later time.