Oracle Exadata is a big step forward from the traditional database server architecture; however, it is still running the standard Oracle Database software. Most of the usual database performance rules still apply, with the addition of some that recognize the advantage of Exadata functionality like Smart Scans, cell join filtering and the flash cache. In this chapter we will cover the Exadata-specific and -related performance topics, metrics, and some relevant internals.

Thankfully Oracle, both at the database layer and in cells, provides lots of performance metrics for our use. However, when looking into any metric, you should know why you are monitoring this and what numbers are you looking for. In other words, how do you know when everything is OK and no action is needed, and when things are bad and action is needed? And in order to avoid wasting effort on fixing the wrong problem, we really need to measure what matters! For database performance, nothing matters more than response time, the actual time the end user (or connected system) has to wait for the response. So if we want to make something faster in the database, we should focus on measuring and then optimizing the response time. All the other metrics and indicators, like the number of I/Os or cache hits, are secondary. End users, who are waiting for their report, care about the time they have to wait only, not secondary metrics like CPU utilization or I/O rate. Nevertheless, often these secondary metrics become very useful for understanding and explaining performance issues.

The key metrics for breaking down database response time are the Oracle wait interface’s wait events. The wait events are discussed in Chapter 10, and we will look into performance monitoring tools that make use of them in the next chapter. However, there are additional useful metrics Exadata provides, such as the number of bytes of data returned by smart scans and the actual amount of I/O avoided thanks to storage indexes, and so on. Such metrics give very important additional info about what’s happening in the database and storage cells during SQL execution. In this chapter we will examine these metrics, and you’ll learn how to get them and what they mean. Even if you are not interested in knowing what each metric means, we still recommend you read this chapter, as it explains some important internals and design decisions behind Exadata.

The information in this chapter should give you a good understanding about some key internal workings of Exadata databases and cells and prepare you for the next chapter, where we will put this knowledge to use when monitoring and troubleshooting Exadata performance.

**Measuring Exadata’s Performance Metrics**

Before we start looking at Exadata-specific performance metrics, let’s examine some internals and review some key elements of Exadata-specific features and metrics. One must understand what the performance numbers actually stand for before trying to monitor or optimize anything with this info.
By now you know that Exadata database nodes don’t do physical disk I/O themselves, but ask the
cells to do the I/O for them. In the case of smart scans, the cell servers will also process the blocks read,
effect their contents, filter rows, and so on. So conceptually, the cells are kind of a black box when
viewed from database layer side. The database layer just requests some blocks of data, and the cells do
the physical I/O work under the hood and return the requested columns of matching rows. Luckily,
Oracle Exadata architects and developers have put decent instrumentation into cells, so the cells keep
track of how much work they have done and importantly, they can also send the metrics back to the
database layer along the results. This allows us—DBAs, developers, and troubleshooters—to have an
end-to-end overview of what happened in the database and the cells when servicing a user’s request or
running a query.

For example, when a query is executed via Smart Scan, you will still see statistics like physical_reads
when you query V$SQLSTATS in the database layer, even though the database layer itself didn’t do any
physical reads directly. Another example is the cell physical I/O bytes saved by storage index
statistic, which is counted in the cell level and not in the database. These numbers are visible in the database
layer thanks to cells sending back useful performance metrics in addition to the data queried.

In addition to the standard Oracle performance tools, we’ll use two custom-built tools in this and
the next chapter. They are more suitable for flexible and advanced performance analysis and allow you
to go beyond the standard wait events and SQL statement level statistics. The first tool, Oracle Session
Snapper, is a script containing just an anonymous PL/SQL block, which measures detailed performance
metrics from V$SESSION, V$SESSION_EVENT, V$SESSION_TIME_MODEL, V$SESSTAT, and so on. The last
performance view, V$SESSTAT, is especially important for advanced performance analysis—it contains
hundreds of dynamic performance counters (over 600 in Oracle 11.2.0.2) for each session in the
instance. In addition to the usual monitoring using wait events, diving into V$SESSTAT gives us a much
better idea of what kind of work Oracle sessions are doing, such as how many I/O requests per second
they are doing, how many full segment scans per second, how many migrated/chained rows had to be
fetched during a scan, and so on.

Revisiting the Prerequisites for Exadata Smart Scans

In this section we will look at the various metrics Oracle Database kernel’s instrumentation provides us.
We will not go into the details of Exadata wait events here, as these are already explained Chapter 10. We
will review how to use these wait events for understanding database performance, and you may find
some of the offloading and Smart Scan material already familiar from Chapter 2, but it’s important to
review some of the concepts here, in the context of monitoring and troubleshooting Exadata
performance.

Because the primary performance booster for data warehousing and reporting workloads is the
Exadata Smart Scan, and for OLTP workloads it is the use of Exadata Smart Flash Cache, we will see first
how to measure whether your workload is benefitting from these features. We will use this knowledge as
building blocks for later database and query performance monitoring.

Exadata Smart Scan Performance

Let’s start with Smart Scan metrics. Before we talk about any metrics, let’s review how the decision to do
a Smart Scan is made in Oracle. Note that a Smart Scan can be used on regular table segments and also
on materialized view segments – which are physically no different from regular tables. Smart Scans can
also be used for full scanning through B*-Tree index segments (index fast full scan) and also bitmap index
segments (bitmap index fast full scan). Scanning through index segments using the “brute force”
multiblock reads approach is very similar to full table scans. The major difference is that inside index
segments there are also index branch blocks, which have to be skipped and ignored, in addition to ASSM