Chapter 4
DmRT for NUMA

In this chapter, the dynamic memory allocator called DmRT for Non-uniform memory access (NUMA) architecture will be discussed. As such, there is no change in strategies, policies, and mechanisms which have been used in DmRT for SMP but one more strategy has been introduced to find out the remote memory.

All the design principals such as strategies, policies, and mechanisms are briefly explained first, then, the method to find out remote memory for NUMA with its results with different test cases has been discussed.

4.1 Design Principals

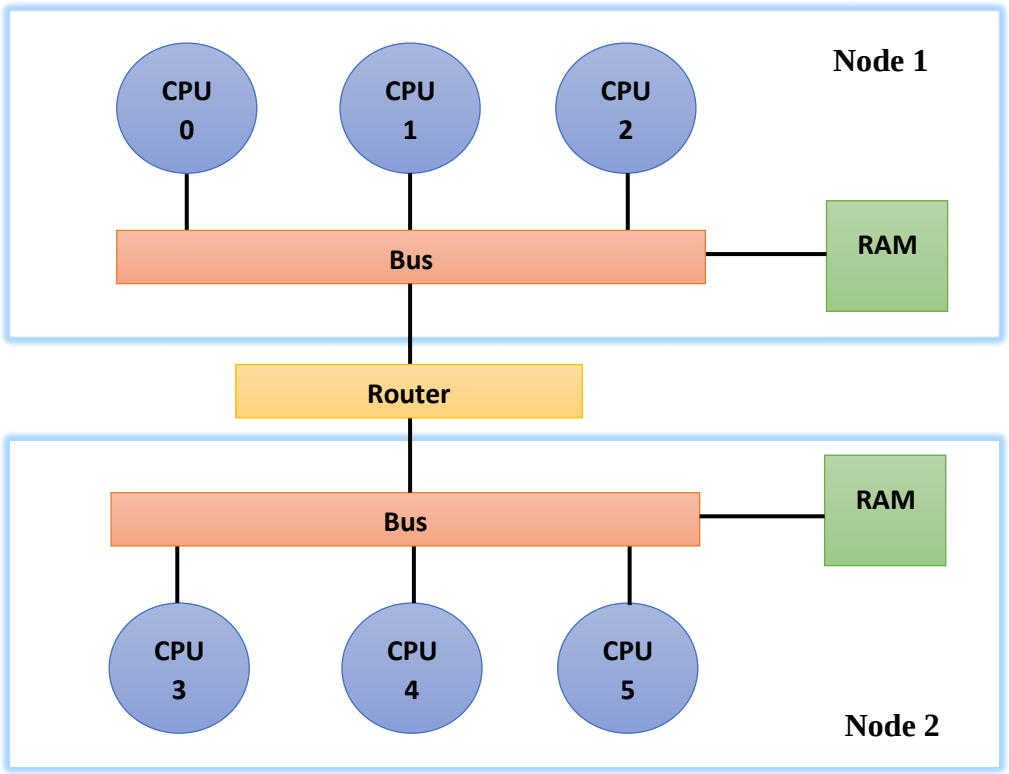

Figure 4.1: NUMA Architecture

NUMA (Non-uniform memory access) is one type of design for computer memory, used in multiprocessing, in which the access time of memory depends on the memory location relative to the processor. In NUMA architecture, a processor can read/write faster from its local memory than the non-local one like the local memory of other processor or shared memory between processors. The benefits of NUMA are limited to particular workloads, mainly on the servers where the data is often associated strongly with certain tasks or users.

In a high-performance computing generation, NUMA is the future of SMP, but its architecture is more complex than the Symmetric multiprocessors. Figure 4.1 shows simple NUMA architecture with two nodes each containing more than one processor and all are sharing a common memory. However, they may have their local memory as well. There are other complex architectures also available which can have 4 or 8 nodes. 4 nodes architecture has been considered for this proposed memory allocator, however, it is very easy to scale it up to 8 nodes.

In this chapter, a dynamic memory allocator DmRT for NUMA has been proposed and its design principals, pseudo code and results with different test cases have been discussed.

4.1.1 Strategy for selecting Remote Memory

The memory allocator which can work on NUMA based architecture for the real-time operating system has been displayed in figure 4.2. There are total four nodes where each node has two processors and each processor within a node are connected with a bus. These all nodes are connected with shared memory with their own local (private) memory.

When any processor requires a memory block, it, first, checks into its local memory; if the required memory block is available, then it will allocate the same block from the local memory. Now, if it fails in finding a local memory, then, it tries to access it from the shared memory. Here, if the memory block is available in shared memory, then it will be allocated. If the block is not found in shared memory also, then, it will ask another processor which is lightly loaded in terms of memory. So, the next step is to find the lightly loaded processor.

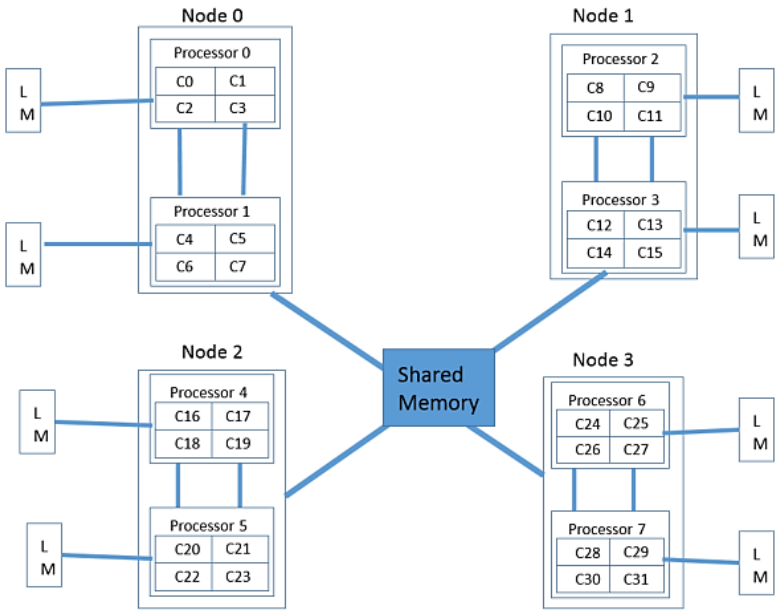

Figure 4.2: Complex NUMA Structure (4 Nodes)

According to memory utilization, each processor can be categorized into four categories as under:

1) Ideal

2) Heavily Loaded

3) Normal Loaded

4) Lightly Loaded

The first step is to calculate the average load for memory utilization of all processors using the following equation [85] [111].

$$Mem_{u\_avg} = \frac{Mem_{u1} + Mem_{u2} + Mem_{u3} + \cdots + Mem_{un}}{n} \quad 4.1$$

The second step is to find the upper and lower threshold value for memory utilization using the following equation [85] [111].

$$T_U = U \times Mem_{u\_avg}$$
$$T_L = L \times Mem_{u\_avg} \quad 4.2$$

Where, T_U = upper limit of threshold,

T_L = lower limit of threshold,

U and L are constants. (U >1 and L< 1)

In the proposed algorithm DmRT, U and L are set to 1.3 and 0.7 respectively, which is interpreted like this. If the memory utilization of a processor is 30% above the Mem_{u_avg}, the processor is said to be heavily loaded and if the memory utilization of a processor is 70% of the Mem_{u_avg}, it is said to be a lightly loaded processor; otherwise, it is considered as a normally loaded processor.

For example, Mem_{u_avg} is 50, then heavily loaded node can be considered as 30 % above Mem_{u_avg} (50 + 30 % of 50 = 65) i.e. its value is 65. And lightly loaded node can be considered as 70 % of Mem_{u_avg} (70% of 50 = 35) i.e. its value is 35.

Hence, Light weight Memory <= 35% Memory utilization

Heavy weight Memory >= 65% Memory utilization

Average (Normal) weight Memory > 35% to < 65% Memory utilization

Ideal Memory < 10% Memory utilization. Ideal Memory is one of the categories for DmRT memory structure in which there is no utilization or memory is only utilized for startup process of processor [85]. The next step is to select the appropriate processor's memory for allocating the memory. Here, which memory will be used to allocate the memory block is decided.

4.1.2 Multiple strategies for different sizes of blocks

As stated earlier, various strategies have been used for allocating the different size of blocks to achieve the benefits of all policies, strategies, and mechanisms.

i. A small block whose size of memory block is < 512 bytes
ii. A normal block whose size of memory block is < threshold (Some predefined size, here, 2Mb)
iii. A large block whose size for request exceeds the threshold or some predefined size

4.1.3 Search Policies and Mechanisms

After defining the strategies, the following policies and mechanisms have been considered to implement these strategies.

I. First, for Small blocks, the best-fit policy is used and implemented using exact-fit mechanisms to reduce the fragmentation in small sizes of blocks due to rounding up the request size of the memory block.

II. Second, for Normal blocks, the good-fit policy is used implemented using segregated lists, which in turn uses an array of unallocated block lists**.**

III. Finally, for Large blocks, the worst-fit policy is used.

4.1.4 Arrangement of blocks

The arrangement of the block is same as discussed in Symmetric Multiprocessing architecture (SMP) in section 3.1.3.

4.2 Pseudo code of Proposed Allocator for NUMA: DmRT

Pseudo code for the arrangement of the block and allocating a memory block is the same as what has been discussed in SMP in section 3.2. But pseudo code to find remote node is mentioned below:

4.2.1 Remote Node Search

$\boldsymbol{Mem_{ui}}$= Memory utilization of i^{th} node

$\boldsymbol{Mem_{us}}$= Memory utilization of self node

$\boldsymbol{T_U}$ = upper limit of threshold,

$\boldsymbol{T_L}$ = lower limit of threshold,

$\boldsymbol{U}$ and $\boldsymbol{L}$ are constants. ($\boldsymbol{U}$>1 and $\boldsymbol{L}$ < 1)

Here $\boldsymbol{U}$ = 1.3 and $\boldsymbol{L}$ = 0.7

$\boldsymbol{Mem_{u\_avg}}$= Average Memory utilization of all available nodes including shared memory

BEGIN

Calculate Memory Utilization of each node

Find Average Memory Utilization.

$$Mem_{u\_avg} = \frac{Mem_{u1} + Mem_{u2} + Mem_{u3} + \cdots + Mem_{un}}{n}$$

Find Upper and Lower Threshold Values

$T_U = H \times Mem_{u\_avg}$

$T_L = L \times Mem_{u\_avg}$

Sort node in ascending order of utilization

Categorize each node Ideal, lightly loaded, Average Loaded and Heavily Loaded

IF Self node is Ideal Node THEN

    Use local memory of self-node for utilization.

ELSE IF Ideal Node is available THEN

    Use local memory of ideal node for utilization

ELSE IF Lightly Loaded Node is available THEN

    IF Memory utilization of Lightly Loaded Node <= Mem_{us} THEN

        Use local memory of Lightly Loaded node for utilization.

    ELSE

        Use local memory of self-node for utilization.

    ENDIF

ELSE IF Average Loaded Node is available THEN

    IF Memory utilization of Average Loaded Node <= Mem_{us} THEN

        Use local memory of Average Loaded node for utilization.

    ELSE

        Use local memory of self-node for utilization.

    ENDIF

ELSE

Use local memory of self-node for utilization.

ENDIF

END

4.3 Results

Here, four different test cases have been considered for NUMA which are mentioned below:

Case 1: Existing allocators from Local and DmRT follows Local → Shared → Ideal

Existing allocators (Dlmalloc, tcmalloc and TLSF) allocate memory block from Local Memory while DmRT, first, tries to allocate a block from Local Memory. If it fails to do so, then, it attempts the same from Shared memory. If it fails, here too, it tries to find ideal memory and allocates a block from it. As DmRT tries to find a memory block from three different types of memory, its execution time will be more than the other allocators, but it provides a consistent execution time. Along with that, it satisfies the maximum number of the requests as well as creates less fragmentation due to proposed allocator (DmRT) structure.

Case 2: Existing allocators from Local and DmRT from Ideal

In this case, all existing allocators allocate memory block from Local memory only, while DmRT, first, finds the ideal memory and it will allocate a memory block from it. Here, existing allocators allocate the blocks from local memory only that is why they have less number of request satisfactions while DmRT has a maximum number of request satisfaction. The other parameters can also perform the best due to its structure.

Case 3: Existing allocators and DmRT both from Ideal

In this case, the existing allocators and DmRT, both first find ideal memory and allocate a block from it. As existing allocators and DmRT, both allocate memory from the ideal memory, the execution time will be almost same but the DmRT will have a maximum number of request satisfied and less fragmentation as an added advantage.

Case 4: Existing allocators and DmRT follow Local → Shared → Ideal

In this case, the existing allocators and DmRT, both first, try to allocate memory block from local and if they fail, they try to allocate the same block from shared memory. If they fail in shared memory too, then, they find ideal memory and allocate same memory block from it. Though both, the existing allocators and DmRT, follow the same path from allocating memory, the proposed allocator, DmRT, defeats all of them from all aspects.

In each case, there are three different test categories have been used.

a. Best case, i.e. the test has been carried out for 100 memory blocks request.
b. Average case, i.e. the test has been carried out for 1000 memory blocks request.
c. Worst case, i.e. the test has been carried out for 2000 memory blocks request.

There are three main parameters which have been considered for the results.

Parameter 1: The execution time should be consistent and minimum.

Parameter 2: Fragmentation should be as low as possible.

Parameter 3: The number of Requests Satisfied should be as high as possible.

For comparisons, the following four memory management algorithms have been used.

a. Dlmalloc
b. tcmalloc
c. TLSF
d. DmRT

All the tests have been carried out on MemSimRT which will be discussed in detail in chapter 5. Each result mentioned here is the average of 100 attempts. The 100 attempts of each case have been mentioned in the Annexure I.

4.3.1 Existing from Local and DmRT follows Local → Shared → Ideal

1. Average of 100 attempts (Best Case, i.e., for 100 memory block requests)

Table 4.1: Existing from Local and DmRT follows Local → Shared → Ideal (Best Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	DmRT
Execution Time (ms)	326.2426	410.8068	290.2026	374.3901
Fragmentation in (%)	43.5037	36.6849	21.5874	10.5141
Request Satisfied in (%)	57.5068	62.3301	76.6088	94.6614

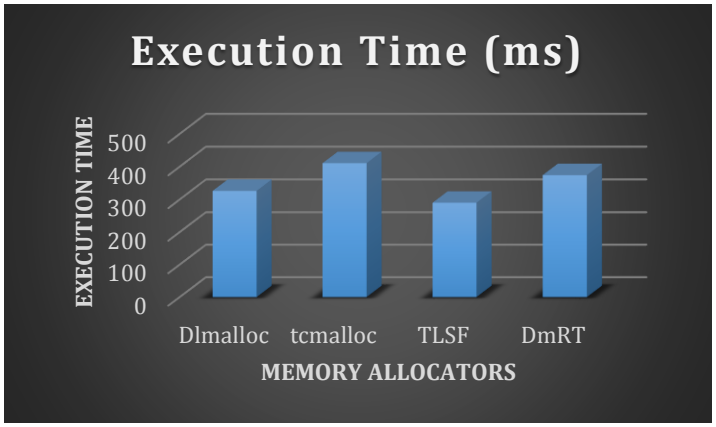

Figure 4.3: Execution time of Memory allocators in Best case

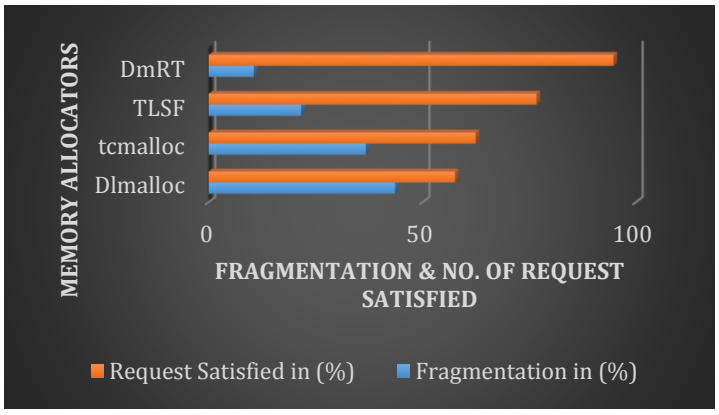

Figure 4.4: Fragmentation & Request Satisfied of Memory allocators in Best case

As shown in Figure 4.1, the DmRT takes **more execution time** than the DLmalloc and the TLSF because the DmRT follows the path of memory allocation from Local to Shared to Ideal, whereas and the existing algorithms allocate from local memory only. In this scenario also, tcmalloc takes maximum execution time.

As shown in figure 4.2, the DmRT **satisfies the maximum requests** and has the **lowest fragmentation** as compared to all other dynamic memory allocators, and Dlmalloc performs exactly opposite to it.

2. Average of 100 attempts (Average Case, i.e., for 1000 memory block requests)

Table 4.2: Existing from Local and DmRT follows Local → Shared → Ideal (Average Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	2013.324	2988.474	1522.335	2303.212
Fragmentation in (%)	52.5491	44.4944	29.5867	15.6241
Request Satisfied in (%)	45.2403	54.2546	65.6095	87.4181

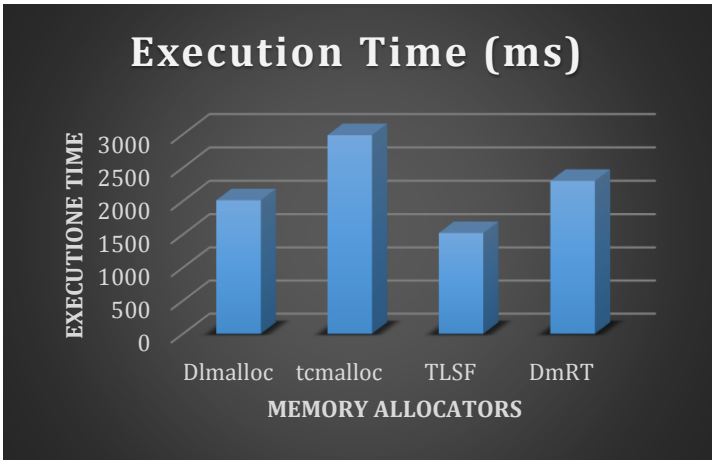

Figure 4.5: Execution time of Memory allocators in Average case

As shown in Figure 4.5, even for 1000 blocks requests, the DmRT takes **more execution time** than the DLmalloc and TLSF, as the DmRT follows the path of memory allocation from Local to shared to Ideal and the existing ones allocate from the local memory only. Here also, tcmalloc takes the maximum execution time.

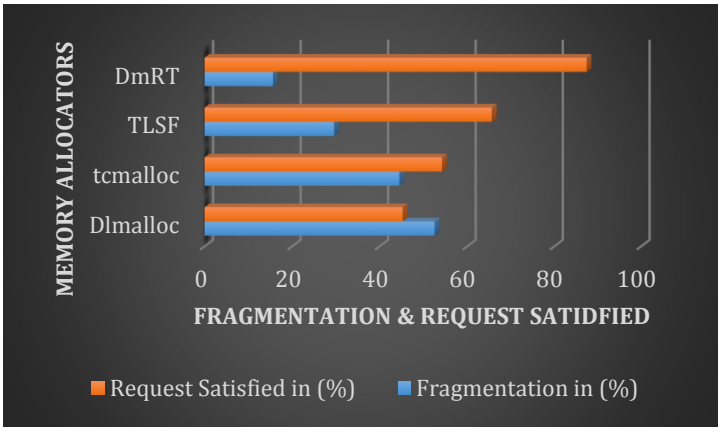

Figure 4.6: Fragmentation & Request Satisfied of Memory allocators in Average case

As shown in figure 4.6, the DmRT **satisfies the maximum requests** and has the **lowest fragmentation** with reference to all other dynamic memory allocators, and Dlmalloc performs exactly opposite to DmRT. Dlmalloc causes a **higher amount of Fragmentation than satisfying number of requests.**

3. Average of 100 attempts (Worst Case, i.e., for 2000 memory block requests)

Table 4.3: Existing from Local and DmRT follows Local → Shared → Ideal (Worst Case)

Algorithms / Parameters	Dlmalloc	Tcmalloc	TLSF	Proposed
Execution Time (ms)	3202.561	4348.651	2049.025	3361.854
Fragmentation in (%)	61.4645	43.3702	36.5828	20.5572
Request Satisfied in (%)	35.006	50.0315	60.055	82.3775

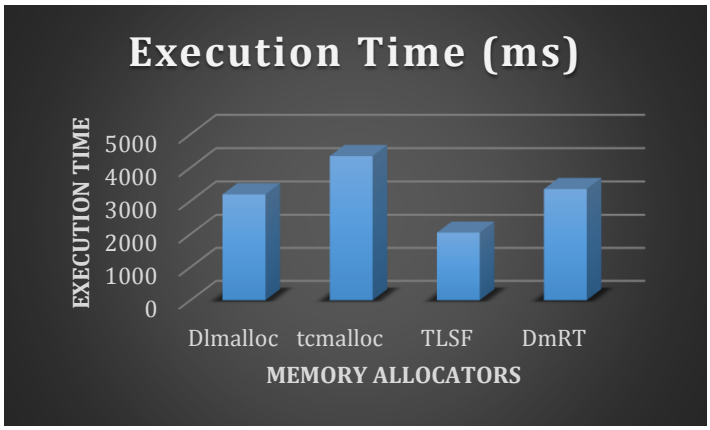

Figure 4.7: Execution time of Memory allocators in Worst case

As shown in Figure 4.7, for 2000 blocks requests also, the DmRT takes **more execution time** than DLmalloc and TLSF. Here too, it follows the same path of memory allocation from Local to shared to Ideal. The existing allocators allocate the same from local memory only yet the tcmalloc takes the maximum execution time.

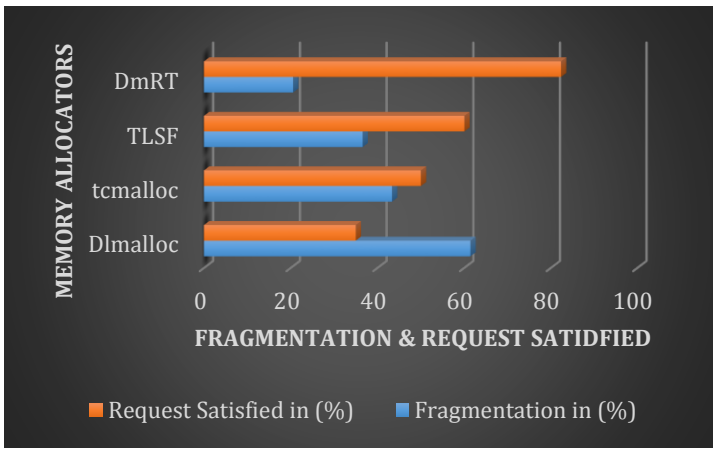

Figure 4.8: Fragmentation & Request Satisfied of Memory allocators in Worst case

As shown in figure 4.8, the DmRT **satisfies maximum requests** and shows the **lowest fragmentation** in comparison with all other dynamic memory allocators. The Dlmalloc is exactly opposite to it causing **higher Fragmentation than satisfying the number of requests.**

4.3.2 Existing From Local and DmRT from Ideal

1. Average of 100 attempts (Best Case, i.e., for 100 memory block requests)

Table 4.4: Existing From Local and DmRT from Ideal (Best Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	338.0589	420.5202	296.4418	245.4583
Fragmentation in (%)	45.2763	33.6326	24.0237	15.4697
Request Satisfied in (%)	57.7885	64.1904	76.9458	89.9526

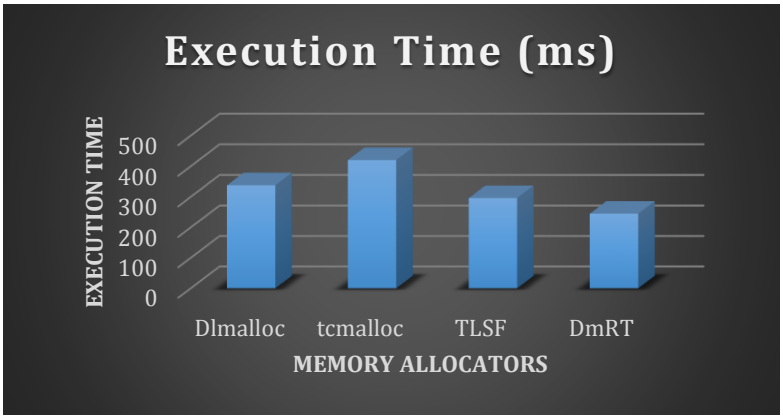

Figure 4.9: Execution time of Memory allocators in Best case

As shown in figure 4.9, the DmRT takes **minimum execution time as** compared to all other dynamic memory allocators, whereas tcmalloc takes the maximum execution time.

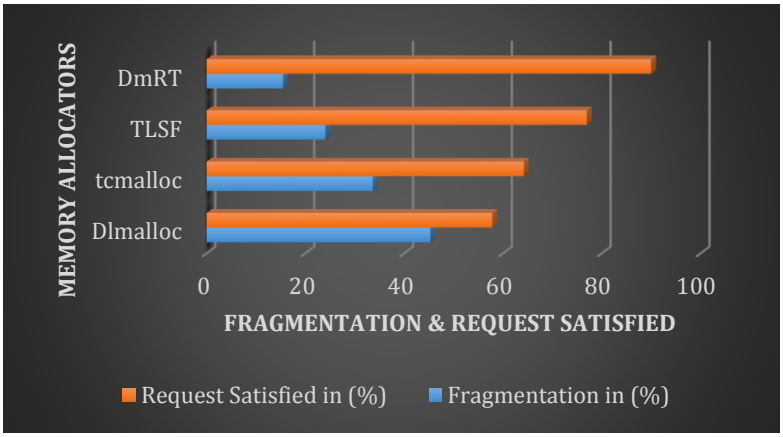

Figure 4.10: Fragmentation & Request Satisfied of Memory allocators in Best case

As shown in figure 4.10, the DmRT **satisfies the maximum requests** and has the **lowest fragmentation due to** allocation of memory from Ideal memory as compared to all other dynamic memory allocators. The Dlmalloc has the exactly opposite performance.

2. Average of 100 attempts (Average Case, i.e., for 1000 memory block requests)

Table 4.5: Existing From Local and DmRT from Ideal (Average Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	2054.716	2995.241	1539.964	1115.835
Fragmentation in (%)	61.1009	53.0719	37.7599	26.0615
Request Satisfied in (%)	34.3767	50.9882	61.8621	73.6385

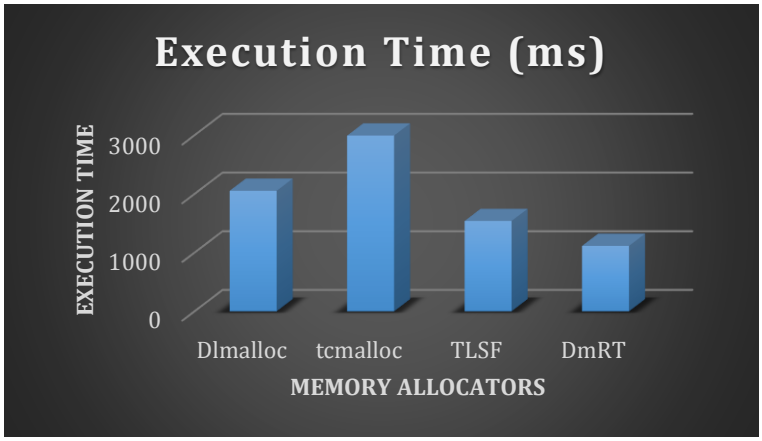

Figure 4.11: Execution time of Memory allocators in Average case

As shown in figure 4.11, the DmRT takes the **minimum execution time among all** dynamic memory allocators, and tcmalloc takes the maximum execution time.

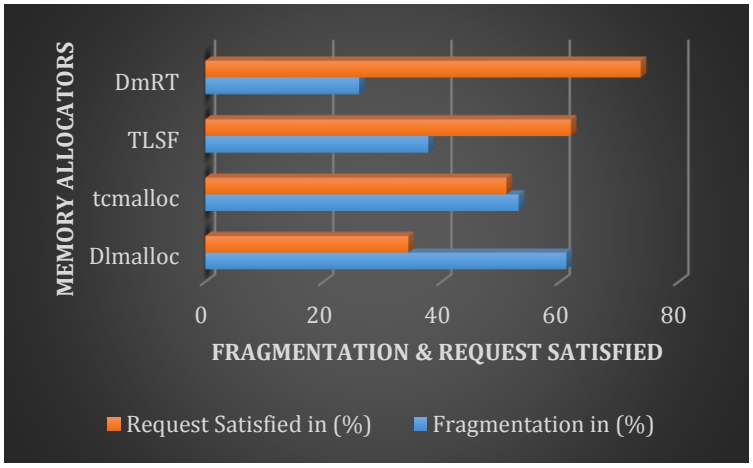

Figure 4.12: Fragmentation & Request Satisfied of Memory allocators in Average case

As shown in figure 4.12, the DmRT **satisfies the maximum requests** and shows the **lowest fragmentation** among all other dynamic memory allocators, and performance point of view, the Dlmalloc is exactly opposite. It causes a **higher amount of Fragmentation than satisfying the number of requests.**

3. Average of 100 attempts (Worst Case, i.e., for 2000 memory block requests)

Table 4.6: Existing From Local and DmRT from Ideal (Worst Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	3283.047	4288.159	2064.704	1785.676
Fragmentation in (%)	60.4389	43.6719	32.0433	26.9948
Request Satisfied in (%)	34.9105	49.962	59.887	74.1195

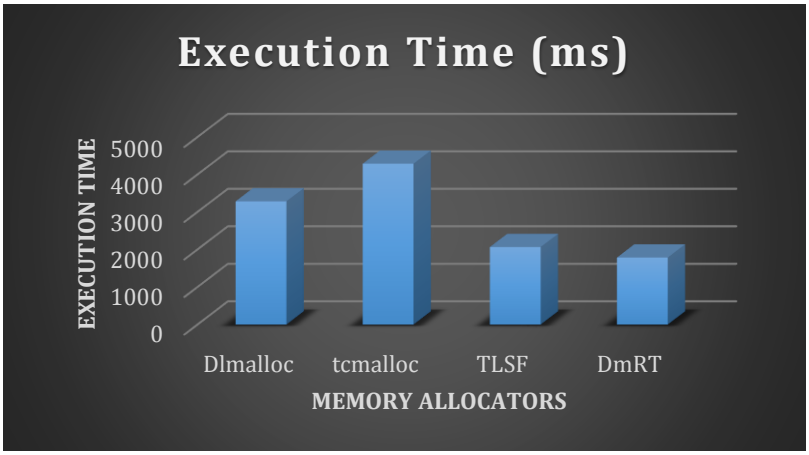

Figure 4.13: Execution time of Memory allocators in Worst case

As shown in figure 4.13, the DmRT takes **minimum execution time as** compared to all other dynamic memory allocators and the tcmalloc takes the maximum execution time.

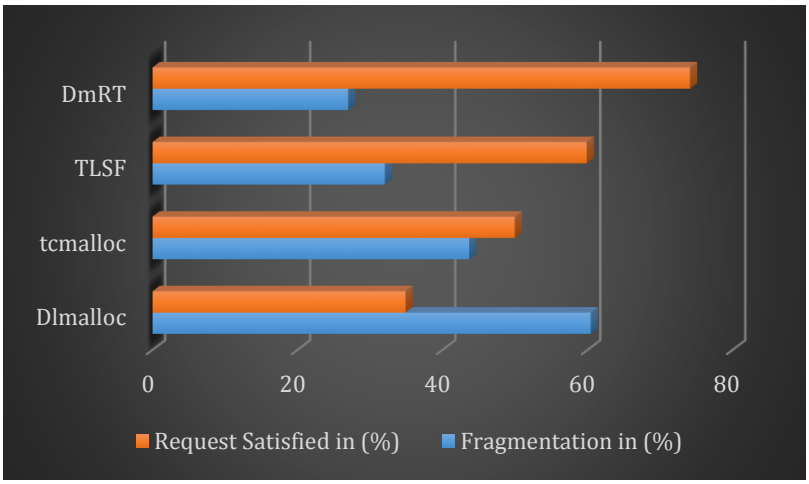

Figure 4.14: Fragmentation & Request Satisfied of Memory allocators in Worst case

As shown in figure 4.14, the DmRT **satisfies the maximum requests** and has the **lowest fragmentation among** all. The Dlmalloc performs exactly opposite to the DmRT with a **higher amount of Fragmentation than satisfying the number of requests.**

4.3.3 Existing and DmRT Both from Ideal

1. Average of 100 attempts (Best Case, i.e., for 100 memory block requests)

Table 4.7: Existing and DmRT Both from Ideal (Best Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	374.8572	444.5905	319.6948	249.566
Fragmentation in (%)	35.2178	26.3902	19.2872	14.5781
Request Satisfied in (%)	67.5358	72.1999	83.1096	89.1851

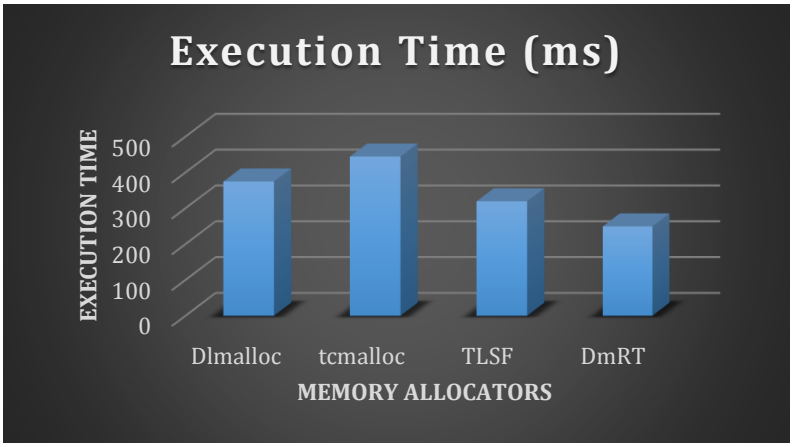

Figure 4.15: Execution time of Memory allocators in Best case

As shown in figure 4.15, the DmRT takes **minimum execution time** as compared to all other dynamic memory allocators, and tcmalloc takes maximum execution time even though all allocators allocate from Ideal memory.

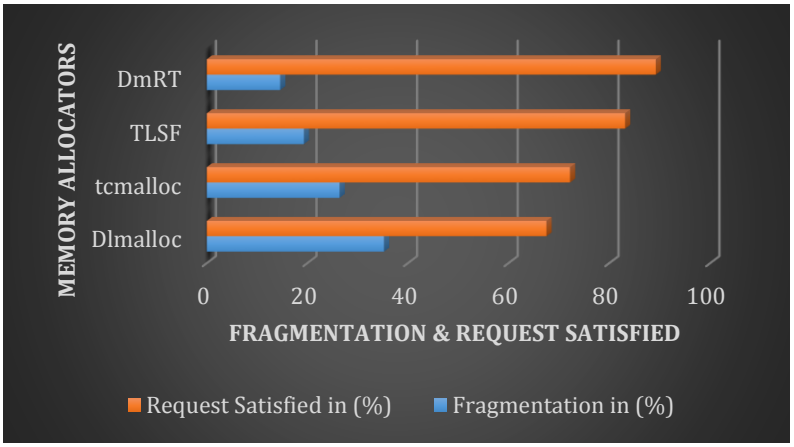

Figure 4.16: Fragmentation & Request Satisfied of Memory allocators in Best case

As shown in figure 4.16, Though all allocators are allocating memory from Ideal memory, the DmRT has the **maximum number of requests satisfied** and has the **lowest fragmentation**. The Dlmalloc is exactly opposite to it, as it has been.

2. Average of 100 attempts (Average Case, i.e., for 1000 memory block requests)

Table 4.8: Existing and DmRT Both from Ideal (Average Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	2110.58	3240.527	1530.277	1149.484
Fragmentation in (%)	43.0248	35.5497	26.3408	19.7854
Request Satisfied in (%)	55.5939	64.722	73.3219	81.1776

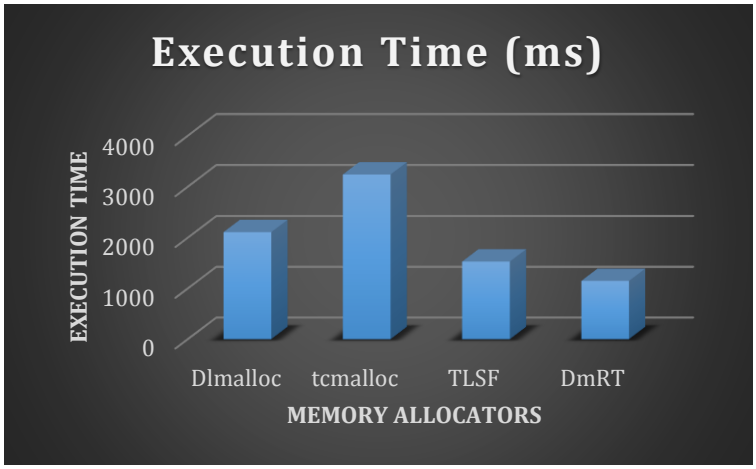

Figure 4.17: Execution time of Memory allocators in Average case

As shown in figure 4.17,

For 1000 blocks request, though all allocators are allocating memory from the Ideal memory, the DmRT takes the **minimum execution time as** compared to all other dynamic memory allocators, and tcmalloc takes the maximum.

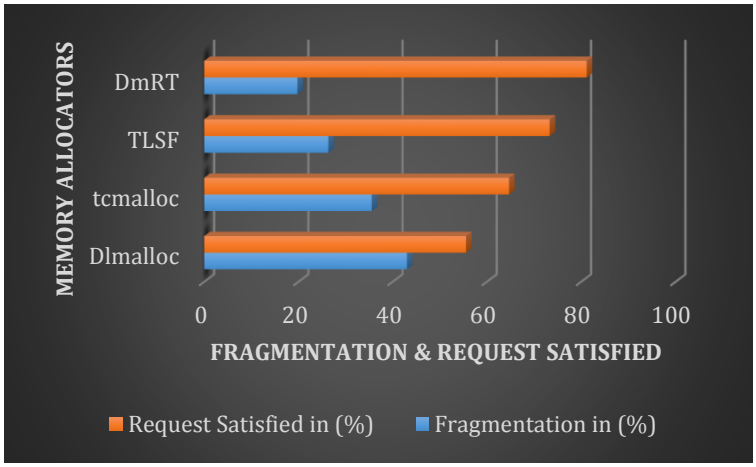

Figure 4.18: Fragmentation & Request Satisfied of Memory allocators in Average case

As shown in figure 4.18,

For 1000 blocks request, though all allocators are allocating memory from the Ideal memory, DmRT, as usual, **satisfies the maximum requests** with the lowest **fragmentation among** all. The Dlmalloc does exactly opposite to it.

3. Average of 100 attempts (Worst Case, i.e., for 2000 memory block requests)

Table 4.9: Existing and DmRT Both from Ideal (Worst Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	3277.428	4467.102	2172.658	1860.321
Fragmentation in (%)	52.6015	43.6602	35.9747	25.1212
Request Satisfied in (%)	44.834	52.813	64.469	74.053

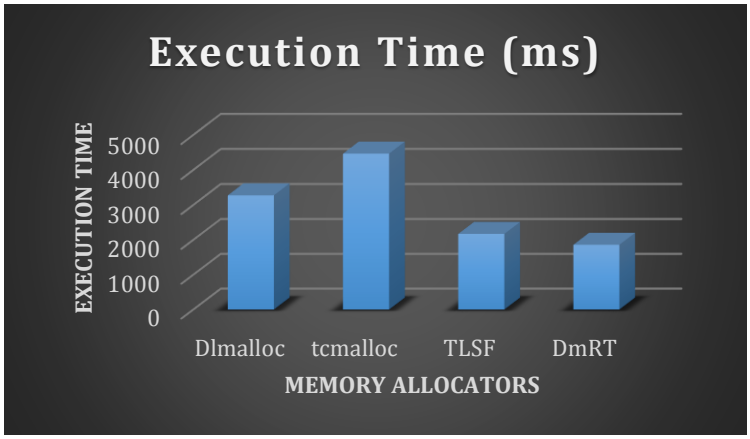

Figure 4.19: Execution time of Memory allocators in Worst case

As shown in figure 4.19, Though all allocators are allocating memory from the Ideal memory, the DmRT takes **minimum execution time among** all and the tcmalloc takes the maximum execution time.

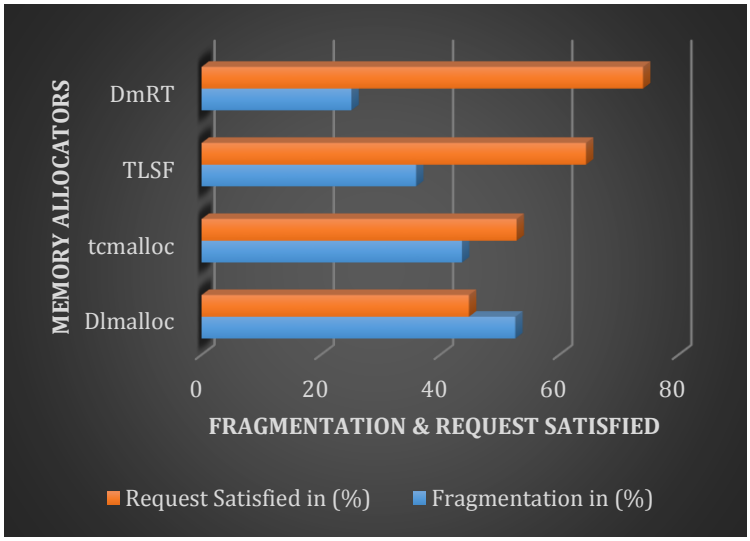

Figure 4.20: Fragmentation & Request Satisfied of Memory allocators in Worst case

As shown in figure 4.20, though all allocators are allocating memory from the Ideal memory, the DmRT has the highest **satisfying requests** and has the **lowest fragmentation**. The Dlmalloc causes a **higher amount of Fragmentation than satisfying the number of requests.**

4.3.4 Existing and DmRT follow Local → Shared → Ideal

1. Average of 100 attempts (Best Case, i.e., for 100 memory block requests)

Table 4.10: Existing and DmRT follow Local → Shared → Ideal (Best Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	528.5204	636.5573	480.8449	385.8492
Fragmentation in (%)	31.4948	23.8764	17.5356	10.4119
Request Satisfied in (%)	71.7431	78.1612	86.8777	93.7361

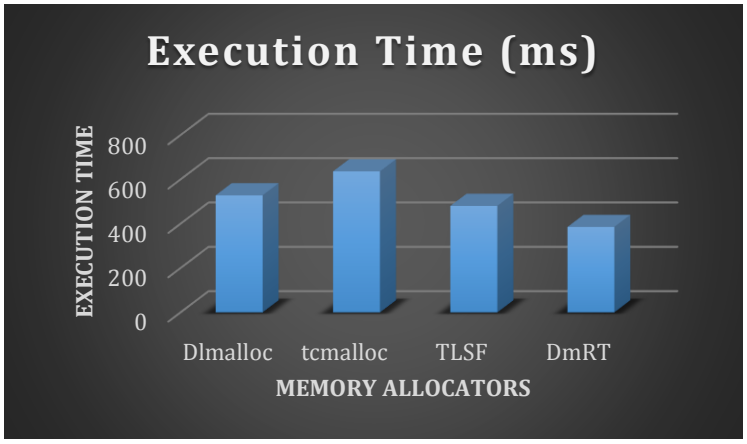

Figure 4.21: Execution time of Memory allocators in Best case

As shown in figure 4.21, Though all allocators following the path of Local, Shared and Ideal memory for allocating memory, DmRT takes **minimum execution time** compared to all other dynamic memory allocators, and tcmalloc takes maximum execution time.

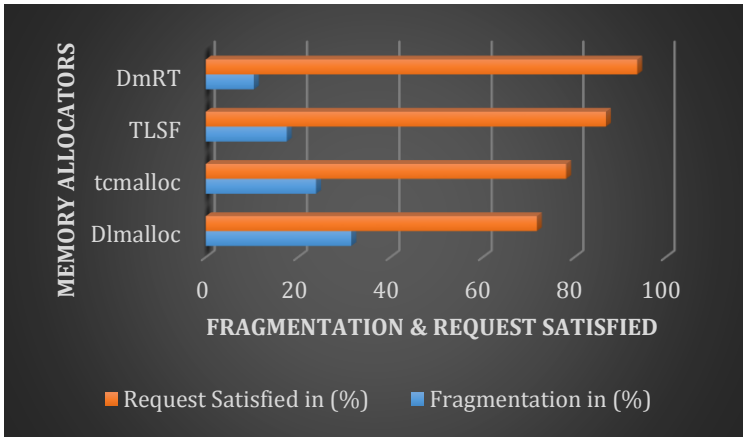

Figure 4.22: Fragmentation & Request Satisfied of Memory allocators in Best case

As shown in figure 4.22, Though all allocators following the path of Local, Shared and Ideal memory for allocating memory, the DmRT **satisfies the maximum requests** with the **lowest fragmentation** and Dlmalloc does exactly opposite to it.

2. Average of 100 attempts (Average Case, i.e., for 1000 memory block requests)

Table 4.11: Existing and DmRT follow Local → Shared → Ideal (Average Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	2928.034	4166.439	2541.517	2252.019
Fragmentation in (%)	38.895	31.6255	22.913	15.0111
Request Satisfied in (%)	62.0389	70.6678	79.9134	87.5092

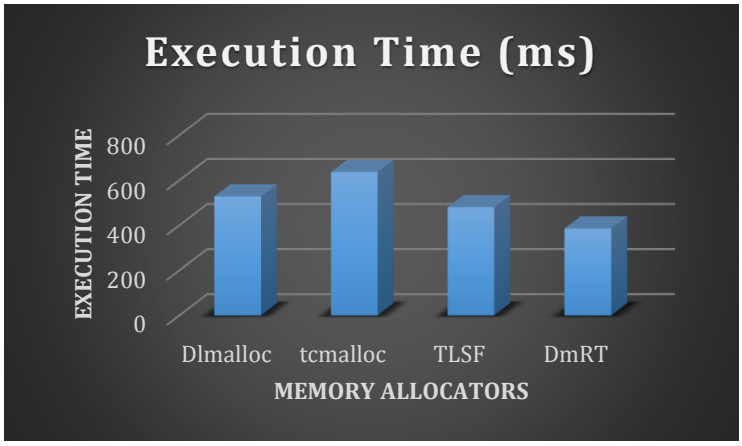

Figure 4.23: Execution time of Memory allocators in Average case

As shown in figure 4.23, for 1000 blocks allocation though all allocators following the path of Local, Shared and Ideal memory, DmRT takes **minimum execution time** as compared to all other dynamic memory allocators, and tcmalloc takes maximum execution time.

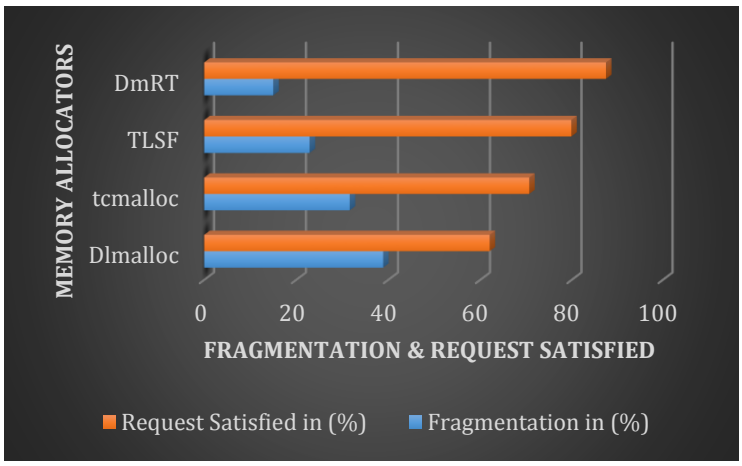

Figure 4.24: Fragmentation & Request Satisfied of Memory allocators in Average case

As shown in figure 4.24, for 1000 blocks allocation though all allocators following the path of Local, Shared and Ideal memory, DmRT **satisfies maximum requests** and has **lowest fragmentation** as compared to all other dynamic memory allocators, and Dlmalloc is exactly opposite to it.

3. Average of 100 attempts (Worst Case, i.e., for 2000 memory block requests)

Table 4.12: Existing and DmRT follow Local → Shared → Ideal (Worst Case)

Algorithms / Parameters	Dlmalloc	tcmalloc	TLSF	Proposed
Execution Time (ms)	4231.555	5107.635	3684.495	3367.702
Fragmentation in (%)	47.3835	38.2598	30.8472	19.1314
Request Satisfied in (%)	50.8095	59.9945	73.883	82.662

Figure 4.25: Execution time of Memory allocators in Worst case

As shown in figure 4.25, for 2000 blocks allocation though all allocators following the path of Local, Shared and Ideal memory, DmRT takes **minimum execution time** as compared to all other dynamic memory allocators, and tcmalloc takes maximum execution time.

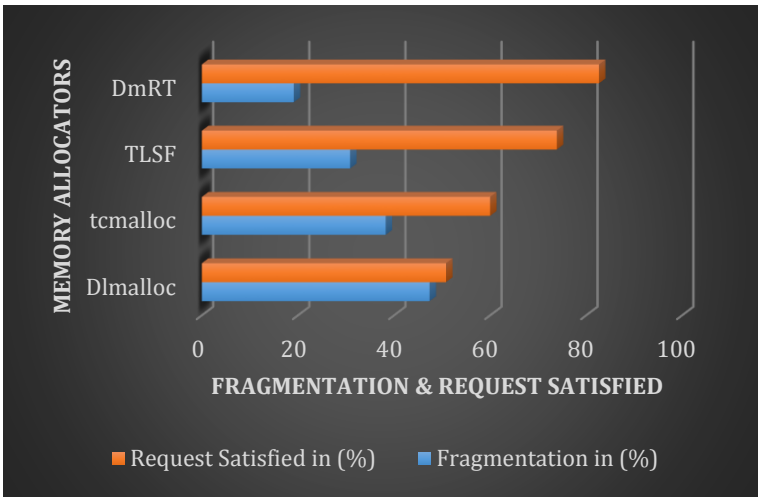

Figure 4.26: Fragmentation & Request Satisfied of Memory allocators in Worst case

As shown in figure 4.26, for 2000 blocks allocation though all allocators following the path of Local, Shared and Ideal memory, DmRT **satisfies maximum request** and has **lowest fragmentation** compare to all other dynamic memory allocators, and Dlmalloc is exactly opposite to it.