Capa

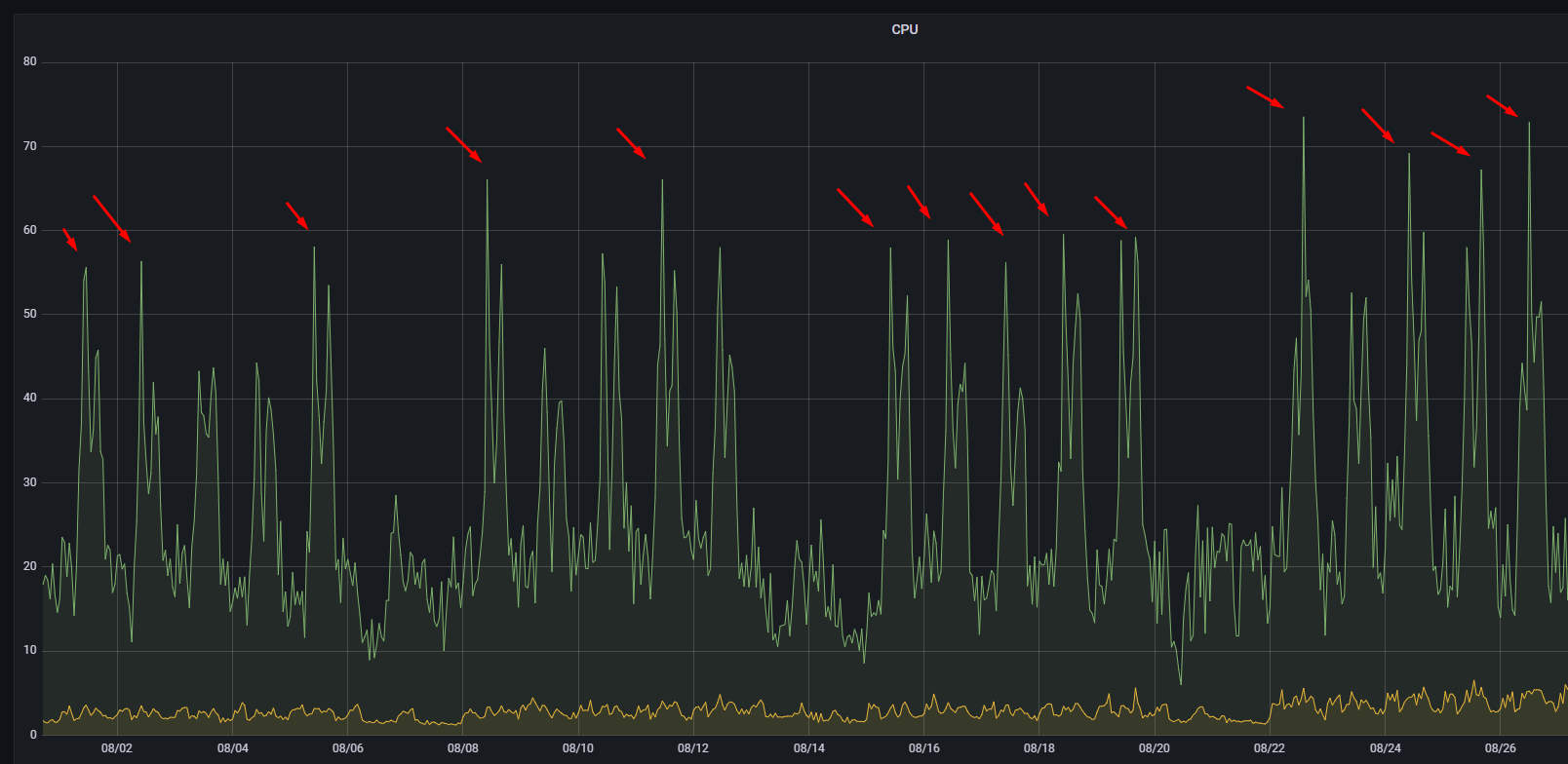
Início da Análise.

Ao longo dos dias um ambiente consideravelmente robusto composto por 64 CPU, divididos em 2 Sockets, estava enfrentando momentos de crises rotineiramente em diversos horários do dia.

O que colocou uma pulga na orelha dos DBA’s de porque estava ocorrendo tal situação.

Poderia ficar aqui explicando e mostrando outros causadores desse desvio de consumo, mas vamos direto ao foco que é mostrar o impacto da configuração de “Cost Threshold of Paralelismo” pode causar no seu ambiente se mau dimensionado.

Logo de cara um Histórico de consumo de CPU desse ambiente nos últimos 30 dias.



Percebam que durante o expediente o consumo de CPU ficava na maior parte acima de 80% de consumo.

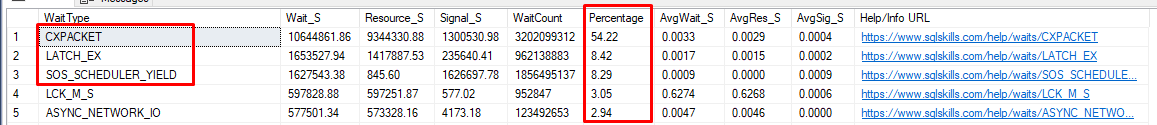
O que por diversas vezes ocasionava Locks e DeadLocks por falta de threads. Afetando diretamente as aplicações que utilizam esse servidor.

Iniciando uma busca do que estava acontecendo no ambiente, procurei os principais WAITS TYPE do SQL Server, e analisar se existia algum entre os tops que poderia estar relacionado a CPU.

E aqui tivemos o primeiro êxito.

No momento da coleta, tínhamos o CXPACKET como TOP 1 Wait do ambiente.

Waits este que está diretamente relacionado a CPU e Paralelismo no SQL Server, e que já nos ajuda a farejar as possíveis causas do consumo desenfreado.



Link com breve explicação sobre CXPACKET: <https://www.sqlskills.com/help/waits/cxpacket/>

Para chegar nessa informação, utilizei uma das queries que encontrei nas *“Googladas da Vida”*

[1 - TopWaits.sql](https://github.com/jcdba/Parallelism/blob/main/1%20-%20TopWaits.sql)

Continuando...

Após encontrar o tipo de Waits, precisaria então verificar a quantidade de queries que estão utilizando Paralelismo para sua solução, e agrupá-las para chegar num denominador correto, para em seguida informar nos parâmetros da instancia SQL Server.

Para isso utilizei o script seguinte:

[2 - Queries\_With\_Parallelism\_x\_Cost.sql](https://github.com/jcdba/Parallelism/blob/main/2%20-%20Queries_With_Parallelism_x_Cost.sql)

Que me trouxe a informação abaixo.

|  |
| --- |
|  |

Em resumo, 2341 Queries com custo abaixo de 10. (já vou explicar o porquê deste corte).

E 1.227 Queries com custo acima de 10, o que representa basicamente 34,38% das queries. Porém, são os 34,38% da fatia mais custosa e onerosa das queries. Pois tem seus custos acima de 10.

E porque estou enfatizando e dividi o resultado no número “10”. Simplesmente porque esse é o Parâmetro que estava fixado na configuração da instancia SQL Server.

Sendo assim, é o que o SQL Server utiliza para determinar queries que serão resolvidas com Paralelismo e consequentemente CPU.

|  |
| --- |
|  |

Veja que além do Cost Threshold for Parallelism (CTFP), temos definido também o Max Degree of Parallelism (MAXDOP). Que também deve ser levado em conta na análise, já que é o número de Core que o SQL vai utilizar para resolver as queries com Paralelismo.

Pois bem, já sabemos o WAITS e a quantidade de Query x Custo que temos, o que fazer?

Nesse caso, defini que o custo de corte seria 80, ou seja, somente queries com CTFP acima de 80 utilizaria de recursos de Paralelismo.

Antes de mostrar o resultado, vou mostrar aqui um Plano de execução de uma query simples e seu custo, no qual estaria acima de 10 e utilizaria recursos de CPU para Paralelismo.

|  |
| --- |
|  |

Percebam que praticamente todo o plano passa por paralelismo, e o custo é de 20,57. O que é relativamente baixo, sendo desnecessário a utilização de CPU para a solução da Query.

Ressalto que essa query foi demasiadamente disparada no ambiente e ocasionou um Pico de 100% de CPU, sim, essa query com custo 20. 😊

Pois bem, como já tinha a análise toda pronta, fizemos a alteração do CTFP de 10 para 80 a quente, já que seu COMMIT e ROLLBACK é instantâneo. E reparem na descida de consumo de CPU.

|  |
| --- |
|  |

Instantaneamente tivemos uma descida de 100% para 60%, e descendo mais ainda até 30% onde se manteve a média.

Gritante não é mesmo? Sim!

Com apenas um ajuste simples e sem precisar de Restart da Instancia ou Indisponibilidade de qualquer aplicação.

Agora vamos olhar o Plano de execução da mesma Query.

|  |
| --- |
|  |

Pois é, somente um Constant Scan e sem utiliza paralelismo. Com certeza nas próximas execuções será resolvida em memória, deixando a query ainda mais rápida.

Para essa alteração surgem algumas dúvidas, que acho interessante pontuar aqui, são elas:

**- A aplicação pode ficar mais lenta, já que não está utilizando CPU (recursos robustos do servidor) para resolver as queries?**

**- Não, o SQL funciona justamente ao contrário, se a query já está me memória, não precisa resolver novamente, pois já tem seu Plano de execução (Hash) pronto para reutilizá-lo.**

**- Se em memória são melhores, não devemos aumentar o parâmetro para não resolver queries com CPU?**

**- Não, precisamos da CPU justamente para resolver soluções aritméticas e queries extremamente custosas que não ficam no buffer de memória. Por exemplo aquelas que contêm CONVERT IMPLICIT (comparação de datatypes diferentes na query).**

**- Essa alteração é vitalícia?**

**- Não, é interessante que seja monitorado e ocorrendo desvios rotineiros de CPU, seja feita uma nova análise das queries e seus custos para definir um novo parâmetro.**

**- Não vai mais ocorrer picos de CPU?**

**- Podem ocorrer picos de CPU sim, se no ambiente existem outras operações que podem apresentar alto consumo de CPU. (por exemplo: Replication, Convert Implicit). Porém, os desvios serão menores, ocorrerão com menos frequência e será mais fácil de apontar um possível responsável pelo desvio.**

Por fim, apesar da rápida implementação e impacto positivo. Essa é uma alteração que requer a análise de um DBA Sênior Especialista.

Por fim, espero ajudar com essa Análise.