Aquí tienes una versión mucho más completa y detallada de los apuntes, estructurada y “cuadriculada” para facilitar la memorización visual. Estos apuntes abarcan todos los aspectos vistos en clase y en la bibliografía, integrando ejemplos, fórmulas y diagramas conceptuales.

APUNTES COMPLETOS: MEDICIÓN DEL RENDIMIENTO DE UN SERVIDOR

II. CONCEPTOS FUNDAMENTALES

1. Carga y Actividad del Servidor

- **Carga (Workload):**
  - **Definición:** Conjunto de tareas y peticiones que demanda el servidor.
  - **Ejemplo:** Usuarios concurrentes, procesos en ejecución, solicitudes HTTP, etc.
- **Actividad del Servidor:**
  - **Definición:** Conjunto de operaciones que se realizan en respuesta a la carga.
  - **Variables a monitorear:**
    - **CPU:**
      - Utilización (porcentaje de tiempo ocupado).
      - Número de procesos en cola (run queue).
      - Frecuencia y fallos de caché.
    - **DRAM:**
      - Uso de memoria, memoria libre, ancho de banda, latencias y fallos de página.
    - **Almacenamiento (HDD/SSD):**
      - Utilización, ancho de banda, latencias, tamaño de colas.
    - **Red:**
      - Utilización, ancho de banda, retransmisiones, errores.
    - **Sistema Global:**
      - Número de usuarios conectados y peticiones totales.

> **Nota:**
> *Un servidor no es “bueno” o “malo” en abstracto; su rendimiento depende de cómo se adapta a un tipo concreto de carga.*

2. Monitor de Actividad

- **Definición:**
  Herramienta o conjunto de herramientas diseñadas para medir, procesar, almacenar y/o visualizar la actividad de un sistema informático.
- **Funciones Típicas:**
  - **Medición:** Captura de variables (CPU, memoria, etc.).

    - **Procesamiento:** Filtrado y análisis estadístico de los datos.
    - **Visualización/Almacenamiento:** Representación en tiempo real o guardado para análisis histórico.

- **Beneficios:**

    - **Para Administradores/Ingenieros:**
        - *Capacity planning*: Predicción de la evolución de la carga.
        - Detección de cuellos de botella y necesidades de reconfiguración.
    - **Para Programadores:**
        - Identificar "hot spots" en la aplicación.
    - **Para el Sistema Operativo:**
        - Ajustes dinámicos en respuesta a la carga.

III. TIPOS DE MONITOREO Y CLASIFICACIONES

1. Según el Momento de Medición

- **Monitor por Eventos:**
    - Mide cada vez que ocurre un evento específico (ej.: apertura de fichero, fallo de caché).
- **Monitor por Muestreo:**
    - Toma medidas a intervalos fijos (cada **T** segundos).
    - **Ejemplo:** Un monitor que se activa cada 5 segundos para recoger datos.

2. Según el Método de Medición

- **Software:**
    - Programas instalados que recogen y procesan la información.
    - **Ejemplos:** `ps`, `top`, `vmstat`, `sar`.
- **Hardware:**
    - Dispositivos físicos integrados en el equipo para medir y visualizar datos.

3. Según la Interacción con el Administrador

- **Batch (Por Lotes):**
    - La recogida se realiza en segundo plano sin interacción directa en tiempo real.
- **Online (Interactivo):**
    - Permite visualizar y, a veces, modificar parámetros en tiempo real (representaciones gráficas, interfaces interactivas).

4. Atributos Clave del Monitor

Atributo	Descripción
Anchura de Entrada	Cantidad de información (número de bytes) que se almacena por cada medida.
Sobrecarga (Overhead)	Recursos (CPU, memoria, etc.) que utiliza el monitor y que pueden afectar el rendimiento del propio sistema.

Atributo	Descripción
Exactitud (Accuracy)	Diferencia (offset) entre el valor medido y el valor real.
Precisión	Consistencia de las mediciones al repetir el proceso (dispersión mínima en resultados).
Resolución	Mínima variación en el valor que el sensor es capaz de detectar.

> **Ejemplo de Sobrecarga:**
> Si un monitor se activa cada 5 s y cada activación consume 6 ms de CPU, la sobrecarga es:
> (\frac{6,\text{ms}}{5000,\text{ms}} \times 100 \approx 0.12%)

IV. MONITORIZACIÓN A NIVEL DE SISTEMA

1. El Directorio `/proc` en Linux

- **Función:**
  Es un sistema de archivos virtual que ofrece acceso a datos internos del S.O.
- **Información Disponible:**
  - Datos globales: `loadavg`, `uptime`, `cpuinfo`, `meminfo`, `net`, etc.
  - Datos específicos por proceso: `/proc/[pid]/stat`, `/proc/[pid]/status`, `/proc/[pid]/cmdline`, etc.
  - Parámetros del kernel: `/proc/sys`.

2. Herramientas y Comandos Esenciales

A. uptime

- **Función:**
  Muestra la hora actual, tiempo de actividad, número de usuarios y la carga media (último 1, 5 y 15 min).
- **Ejemplo de Salida:**

```
% uptime
 1:21pm up 1 day, 4:09, 18 users, load average: 1.04, 0.30, 0.09
```

B. ps

- **Función:**
  Lista los procesos en ejecución con detalles como %CPU, %MEM, estado, etc.
- **Ejemplo de Salida:**

```
% ps aur
USER     PID  %CPU %MEM  VSZ  RSS TTY      STAT START   TIME COMMAND
miguel 29951 55.9  0.1 1448 384 pts/0    R    09:16  0:11 tetris
```

C. top

- **Función:**
  Monitoriza en tiempo real procesos, uso de CPU, memoria, etc.
- **Aspectos Destacados:**
  - Resumen de carga, tareas (running, sleeping, etc.).
  - Desglose de la utilización de CPU (us, sy, ni, id, wa, hi, si, st).

D. vmstat

- **Función:**
  Muestra estadísticas sobre procesos, memoria, paginación, E/S, y CPU.
- **Ejemplo de Uso:**

```
% vmstat 1 6
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-
----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
st
 0  0    868   8964  60140 342748    0    0    23     7  222  199  1  4 80 15
0
```

E. sar

- **Función:**
  Herramienta flexible para la recogida y análisis de estadísticas del sistema, tanto en tiempo real como históricas.
- **Modos de Uso:**
  - **Interactivo:** Muestra datos en pantalla a intervalos (ej. `sar -P ALL 1` para CPU).
  - **Histórico:** Lee archivos binarios (guardados en `/var/log/sysstat/saDD`).
- **Ejemplo Básico:**

```
% sar -u
 00:00:00 CPU   %usr   %nice   %sys   %wa   %st   %idle
 00:05:00 all    0.09    0.00    0.08   0.00   0.00  99.83
```

3. Monitorización Específica de Recursos

A. Almacenamiento (Discos)

- **Comando:** `sar -d`
- **Indicadores:**
  - **tps:** Transacciones por segundo.
  - **rd_sec/s, wr_sec/s:** Sectores leídos/escritos por segundo.

- **avgrq-sz, avgqu-sz, await, svctm, %util:** Tamaño de solicitudes, cola, tiempos y utilización.

- **Ejemplo:**

```
% sar -d 10 2
18:46:09 DEV  tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz  await  svctm
%util
18:46:19 sda  1.70  33.60     0.00     19.76     0.00     0.47  0.47   0.08
```

B. Red

- **Monitoreo con sar:**
  Permite mostrar tráfico, errores, estadísticas por protocolo e interfaz.
- **Ejemplo:**
  Mostrar datos de tráfico TCP y errores cada 1 s.

4. Almacenamiento de Datos Muestreados

- **sadc (System Accounting Data Collector):**
  - Se ejecuta cada 5 minutos (o intervalo configurable) y almacena los datos en archivos binarios diarios.
- **Cálculo de Anchura de Entrada:**
  - Ejemplo: Archivo de 3.049.952 bytes por día, con 288 muestras →
    [\text{Anchura media} \approx \frac{3049952,\text{bytes}}{288} \approx 10590,\text{bytes/muestra}]

V. MONITORIZACIÓN DE SERVIDORES DISTRIBUIDOS

1. Contexto

- Cuando se monitorizan múltiples servidores en red, es más práctico emplear herramientas especializadas.

2. Herramientas Comunes

- **Ejemplos:** Nagios, Naemon, Shinken, Ganglia, Munin, Zabbix, Elastic.
- **Características Deseables:**
  - **Escalabilidad:** Soporte para servidores monitores secundarios, auto-descubrimiento.
  - **Extensibilidad:** Plugins para añadir nuevas métricas y protocolos (SNMP, IPMI, HTTP, JMX).
  - **Prestaciones:** Soporte para polling y pushing, envío por lotes, compresión de datos.
  - **Fiabilidad y Seguridad:** Validación de datos, gestión de alarmas, encriptación y hashing para evitar suplantación.

VI. PROFILING: MONITORIZACIÓN A NIVEL DE APLICACIÓN

1. Objetivos del Profiler

- **Medir el rendimiento de aplicaciones:**
  - Tiempo total de ejecución (*elapsed time*).
  - Tiempo en modo usuario (*user time*) y en modo kernel (*system time*).
- **Detectar "hot spots":**
  - Identificar funciones o secciones críticas que consumen la mayor parte del tiempo.
- **Recoger información adicional:**
  - Número de llamadas a funciones, fallos de página, cambios de contexto (voluntarios e involuntarios).

2. Herramientas de Profiling

A. /usr/bin/time

- **Función:**
  Mide tiempos de ejecución y algunos parámetros de uso de recursos.
- **Ejemplo:**

```
% /usr/bin/time -v ./matr_multiplication
User time (seconds): 10.47
System time (seconds): 0.01
Percent of CPU this job got: 99%
Elapsed (wall clock) time: 0:10.49
Maximum RSS (kbytes): 12688
Major page faults: 0
Minor page faults: 2982
```

B. gprof

- **Características:**
  - Se requiere instrumentar el código (compilar con `-pg`).
  - Proporciona dos salidas principales:
    - **Flat Profile:** Tiempo de ejecución "self" de cada función.
    - **Call Graph:** Relación y frecuencia de llamadas entre funciones.
- **Flujo de Uso:**
  1. Compilar con instrumentación:

     ```
     gcc prog.c -pg -g -o prog
     ```

  2. Ejecutar el programa para generar `gmon.out`.
  3. Analizar con:

     ```
     gprof prog
     ```

- **Ejemplo de Salida (Fragmento):**

- **Flat Profile:**

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  s/call   s/call  name
 45.22   23.61     23.61        1    23.61   52.13   main
 33.98   41.34     17.74        2     8.87   11.24   bucle1
 16.32   49.86      8.52       18     0.47    0.47   bucle3
  4.33   52.13      2.26        1     2.26    6.05   bucle2
```

- **Call Graph:**
  Muestra cómo se interconectan `main`, `bucle1`, `bucle2` y `bucle3` (por ejemplo, cuántas veces `bucle1` llama a `bucle3`).

C. perf

- **Características:**
  - Basado en eventos hardware y software.
  - Permite análisis a nivel de CPU, identificando funciones que consumen más ciclos, fallos de caché, etc.
- **Ejemplos de Comandos:**
  - **Estadísticas generales:**

```
sudo perf stat -e task-clock,cycles,context-switches,page-
faults,instructions,cache-misses -r 3 ./matr_multiplication
```

  - **Generar reporte:**

```
sudo perf report --stdio
```

  - **Análisis a nivel de código:**

```
sudo perf annotate --stdio
```

D. Valgrind

- **Características:**
  - Emula la ejecución del programa, permitiendo detectar errores de memoria y analizar el uso del heap.
  - Alto sobrecoste en tiempo de ejecución.
- **Ejemplo:**

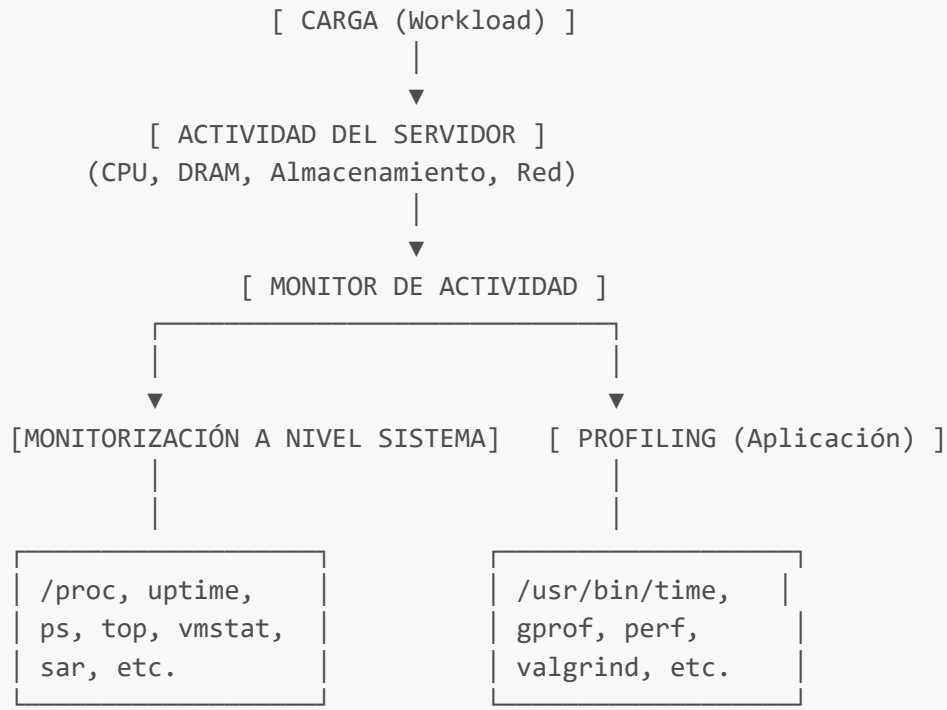
```

```

```
valgrind --tool=memcheck ./programa
```

E. Otros Profilers

- **V-Tune (Intel) y CodeXL (AMD):**
  Herramientas especializadas para análisis en sistemas multicore o GPU.

VII. DIAGRAMA CONCEPTUAL



VIII. CONSIDERACIONES FINALES Y MÉTODO SISTEMÁTICO

1. Conclusiones Clave

- **Adaptabilidad:**
  La eficiencia del servidor depende de cómo se adapta a la carga específica que recibe.
- **Importancia de la Monitorización:**
  Permite:
  - Predecir la evolución de la carga.
  - Detectar y corregir cuellos de botella.
  - Optimizar tanto la infraestructura (hardware) como el software (aplicaciones).
- **Selección de Herramienta:**
  Dependerá del nivel de análisis requerido:
  - A **nivel de sistema:** `/proc`, `uptime`, `ps`, `top`, `vmstat`, `sar`.
  - A **nivel de aplicación (profiling):** `/usr/bin/time`, `gprof`, `perf`, `valgrind`.

2. Método USE para la Monitorización

- **U (Utilization):**
  Fracción del recurso que está siendo utilizado.
- **S (Saturation):**
  Nivel de ocupación de las colas de tareas en espera.
- **E (Errors):**
  Incidencias y errores detectados en el uso de los recursos.

IX. RESUMEN VISUAL PARA MEMORIZACIÓN

- **Carga → Actividad → Monitor**
  [Variables: CPU, memoria, discos, red]
- **Medición:**
  - **Por Evento:** Cada acción relevante.
  - **Por Muestreo:** Cada T segundos.
- **Herramientas:**
  - **Sistema:** uptime, ps, top, vmstat, sar.
  - **Aplicación:** /usr/bin/time, gprof, perf, valgrind.
- **Análisis:**
  - Determinar cuellos de botella y “hot spots”.
  - Utilizar datos históricos y en tiempo real para la toma de decisiones.

Tema 4

I. CARACTERÍSTICAS DE UN BUEN ÍNDICE DE RENDIMIENTO

Un índice de rendimiento adecuado debe cumplir varios requisitos para que sus mediciones sean útiles y comparables:

- **Repetibilidad:**
  - **Definición:** Bajo las mismas condiciones, el índice debe obtener siempre el mismo valor.
  - **Importancia:** Permite comparar resultados sin variabilidad aleatoria.
- **Representatividad y Fiabilidad:**
  - **Definición:** Si el índice de un sistema A es consistentemente mejor que el de B, ello debe reflejar un rendimiento real superior.
  - **Importancia:** Garantiza que la medida refleje fielmente el comportamiento del sistema.
- **Consistencia:**
  - **Definición:** El índice se debe poder medir en cualquier sistema, sin depender de la arquitectura o del sistema operativo.

- **Importancia:** Facilita comparaciones entre equipos muy diferentes.

- **Facilidad de Medición:**
  - **Definición:** Las variables necesarias para obtener el índice deben ser fáciles y directas de medir.
  - **Importancia:** Reduce la complejidad y el coste de la evaluación.

- **Linealidad:**
  - **Definición:** Un incremento en el índice debería corresponder a un incremento proporcional en el rendimiento real.
  - **Importancia:** Permite interpretar los cambios de forma directa.

II. BÚSQUEDA DE UN BUEN ÍNDICE DE RENDIMIENTO PARA LA CPU

1. Fórmula Básica del Tiempo de Ejecución

El tiempo de ejecución de un programa ((T_{ejecución})) se puede expresar como:

[T_{ejecución} = N \times \text{CPI} \times T_{ciclo} = \frac{N \times \text{CPI}}{f_{\text{reloj}}}]

Donde:

- (N) = número de instrucciones.
- **CPI** = número medio de ciclos por instrucción.
- (T_{ciclo}) = tiempo de un ciclo (inverso de la frecuencia).
- (f_{\text{reloj}}) = frecuencia del reloj.

2. Por Qué No Son Índices Suficientes...

- **Frecuencia de Reloj ((f_{\text{reloj}})) y CPI:**
  - **Problema:** Un procesador con menor (f_{\text{reloj}}) o peor CPI puede rendir mejor si optimiza otros aspectos (por ejemplo, la ejecución paralela o la arquitectura interna).

- **MIPS (Mega Instrucciones Por Segundo):**
  - **Fórmula:**
    [\text{MIPS} = \frac{N}{T_{ejecución}} \times 10^6 = \frac{f_{\text{reloj}}}{\text{CPI} \times 10^6}]
  - **Problemas:**
    - Varían entre distintos programas incluso en el mismo equipo.
    - Un procesador con mayor MIPS podría ejecutar un programa más lentamente si requiere más instrucciones para lograr la misma tarea (por ejemplo, diferencias entre arquitecturas RISC y CISC).

- **MFLOPS (Mega Floating-point Operations Per Second):**
  - **Aspectos a considerar:**
    1. No todas las operaciones en coma flotante tienen la misma complejidad.
    2. Los formatos numéricos pueden variar entre arquitecturas, afectando la exactitud.
    3. No se pueden usar exclusivamente para evaluar el rendimiento global de la CPU.

III. CONCLUSIÓN DEL ANÁLISIS DE ÍNDICES

- **No existe un índice perfecto:**
  Ningún índice cumple por sí solo todos los requisitos (repetibilidad, representatividad, consistencia, facilidad y linealidad).

- **Uso del Tiempo de Ejecución:**
  Se puede medir el rendimiento a través del tiempo que tarda en ejecutarse un programa.

  - **Ventajas:**
    - Si el programa está escrito en un lenguaje de alto nivel y se ejecuta en un entorno controlado, se cumple la repetibilidad y la consistencia.
  - **Desventaja:**
    - La comparación es válida solo para ese programa, por lo que no se puede generalizar a todas las aplicaciones.

- **La Importancia de la Carga Utilizada:**

  - La comparación entre servidores o CPUs solo es válida respecto a la carga (workload) de prueba empleada.
  - **Problema de la carga real:**
    - **Variabilidad:** Cambia a lo largo del tiempo.
    - **Reproducibilidad:** Difícil de reproducir exactamente, ya que interactúa con el propio sistema (un servidor rápido puede recibir más solicitudes).

- **Solución:**

  - Utilizar **modelos de carga** (workload models) que sean representativos y reproducibles para realizar comparaciones.

IV. MODELO DE CARGA Y ESTRATEGIAS DE MEDICIÓN

1. Modelado de la Carga

- **Carga Real vs. Modelo de Carga:**

  - **Carga Real:**
    - Es la carga que efectivamente recibe el sistema, pero es variable y difícil de reproducir.
  - **Modelo de Carga:**
    - Una representación simplificada y estadística de la carga real, que permita reproducir condiciones de prueba de forma controlada.

- **Características de un Buen Modelo de Carga:**

  - **Representatividad:** Debe reflejar lo más fielmente posible la carga real del sistema.
  - **Simplicidad:** Debe ser compacto y permitir tiempos de medición razonables.

2. Estrategias para Obtener Modelos de Carga

- **Caracterización de la Carga Real:**
    - **Pasos:**
        1. **Identificar los Recursos Clave:** CPU, memoria, discos, red, etc.
        2. **Seleccionar Parámetros Característicos:** Por ejemplo, % de utilización de CPU, número de lecturas/escrituras, accesos a la red, etc.
        3. **Medir con Monitores de Actividad:** Utilizar herramientas de muestreo (como sar, vmstat, etc.).
        4. **Analizar los Datos:** Usar medias, histogramas, clustering, etc.
        5. **Generar el Modelo:** Seleccionar representaciones de la carga (por ejemplo, solicitudes típicas) y distribuirlas temporalmente según los datos recolectados.
- **Benchmarking o Referenciación:**
    - **Definición:** Utilización de programas estándar para comparar el rendimiento entre equipos.
    - **Componentes Clave de un Benchmark:**
        1. **Test Workload:** La carga de prueba que se aplicará al sistema.
        2. **Reglas de Ejecución:** Procedimientos y condiciones a seguir para asegurar comparabilidad.
        3. **Métrica (Índice de Rendimiento):** El valor numérico que resume el rendimiento (por ejemplo, transacciones por segundo, QphH@Size, etc.).

V. TIPOS DE BENCHMARKS

1. Según la Estrategia de Medida

- **Benchmarks Basados en Tiempo Fijo:**
    - Miden el tiempo para ejecutar un número determinado de tareas.
- **Benchmarks Basados en Tareas Fijas:**
    - Miden la cantidad de tareas que se pueden ejecutar en un tiempo preestablecido.
- **Ejemplos Específicos:**
    - **SLALOM:** Evalúa la exactitud de la solución alcanzada en 1 minuto.
    - **TPC-C:** Mide el número de transacciones (consultas, pagos, nuevos pedidos, etc.) procesadas por segundo, exigiendo tiempos de respuesta específicos.

2. Según la Generalidad del Test

- **Microbenchmarks:**
    - Evalúan componentes o grupos concretos (por ejemplo, CPU, caché, memoria, discos, red).
    - **Ejemplos:**
        - **Whetstone (1976):** Opera con cálculos en coma flotante usando operaciones simples.
        - **Linpack (1983):** Resuelve sistemas densos de ecuaciones lineales; se utiliza para listar los supercomputadores.
        - **Dhrystone (1984):** Mide operaciones con enteros (copias, comparaciones de cadenas).
- **Macrobenchmarks:**
    - Evalúan el rendimiento del sistema completo o de aplicaciones reales.

- **Ejemplos:**
  - Benchmarks de servidores web, de bases de datos, o de sistemas ofimáticos.

VI. EJEMPLOS DE MICROBENCHMARKS Y HERRAMIENTAS ADICIONALES

- **Otros Microbenchmarks y Herramientas:**
  - **Stream:** Mide el ancho de banda de la memoria DRAM y cachés.
  - **IOzone:** Evalúa el rendimiento del sistema de ficheros (lectura/escritura de discos).
  - **Iperf:** Mide el rendimiento de conexiones TCP/UDP.
  - **Conjuntos de pruebas integradas:**
    - **AIDA64, Sandra** (Windows)
    - **Phoronix Test Suite:** Multiplataforma, con generación de informes automáticos y recopilación de datos de sensores.

VII. BENCHMARK SPEC CPU 2017

1. Introducción a SPEC

- **SPEC (Standard Performance Evaluation Corporation):**
  - Organización sin ánimo de lucro que establece benchmarks para evaluar el rendimiento y la eficiencia energética.
- **Conjuntos en SPEC CPU 2017:**
  - **SPECspeed®2017:**
    - *Integer* (operaciones enteras)
    - *Floating Point* (operaciones en coma flotante)
  - **SPECrate®2017:**
    - Para medir la tasa de procesamiento (cuántos programas se pueden ejecutar en un tiempo dado).

2. Conceptos de “Base” vs. “Peak”

- **Base:**
  - Compilación con opciones conservadoras y uniformes para todos los programas.
- **Peak:**
  - Permite optimizar individualmente cada programa para obtener el máximo rendimiento.

3. Cálculo del Índice SPEC

- **Procedimiento:**
  - Cada programa del benchmark se ejecuta 3 veces y se selecciona el resultado intermedio.
  - El índice SPEC es la media geométrica de los ratios entre el tiempo de referencia (tREF) y el tiempo medido (tbase) en cada programa.
- **Ejemplo de Cálculo:**

  Supongamos los siguientes datos para algunos programas:

Benchmark	tREF (s)	tbase (s)	Ratio (tREF/tbase)
600.perlbench_s	1774	358	4,96
602.gcc_s	3981	546	7,29
605.mcf_s	4721	866	5,45
…	…	…	…

El índice SPEC se obtiene calculando la media geométrica de los 10 ratios:

[\text{Índice SPEC} = \sqrt[10]{\text{Ratio}_1 \times \text{Ratio}*2 \times \cdots \times \text{Ratio}*{10}}]

En el ejemplo, el índice SPEC podría ser, por ejemplo, 5,31.

VIII. BENCHMARKS DE SISTEMA COMPLETO

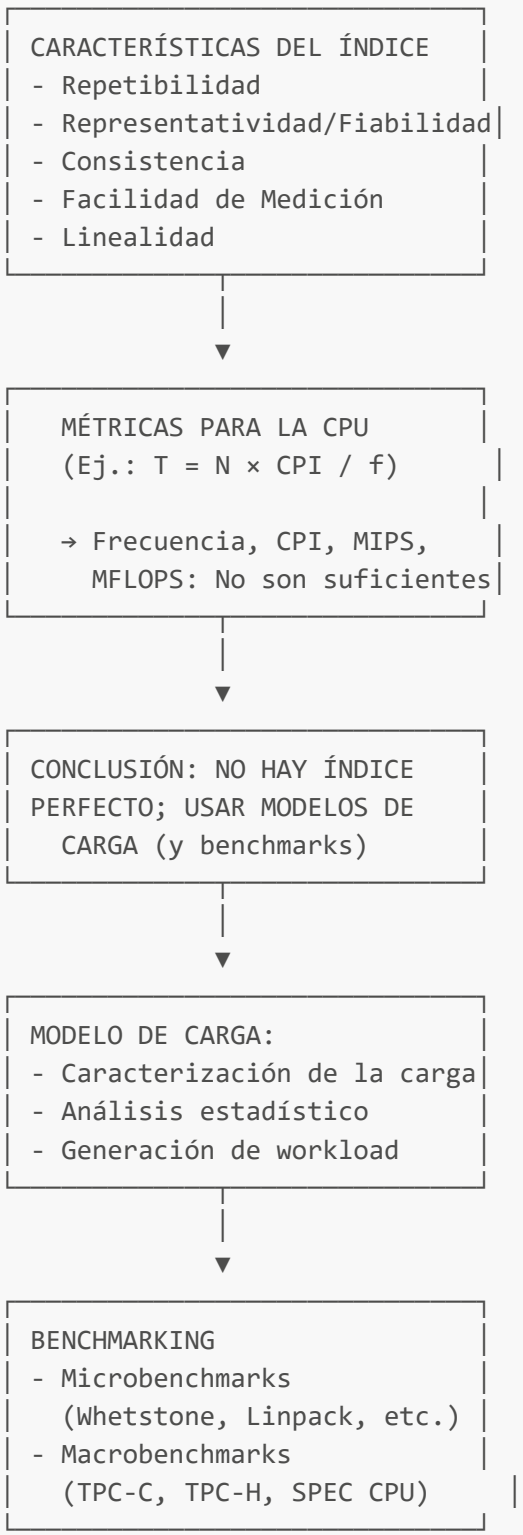
1. Benchmark TPC (Transactions Processing Performance Council)

- **Tipos Principales:**
  - **TPC-C (OLTP):**
    - Simula el procesamiento de transacciones en una gran compañía (por ejemplo, gestión de pedidos, pagos, envíos, etc.).
    - **Métrica:** Número de transacciones procesadas por segundo (con requisitos de tiempos de respuesta, por ejemplo, 90% por debajo de 5 s), además de indicadores de consumo energético y coste por transacción.
  - **TPC-H (Decision Support):**
    - Incluye 22 tipos de consultas complejas sobre grandes volúmenes de datos.
    - **Métrica:** Composite Query-per-Hour Performance Metric (QphH@Size), considerando el tamaño de la base de datos (factor de escala).

2. Otros Benchmarks de Sistema Completo

- **Servidores de Ficheros:**
  - SPECstorage Solution 2020, SPEC SFS2014.
- **Cliente/Servidor Java:**
  - SPECjEnterprise2018 (JEE), SPECjms2007 (JMS), SPECjvm2008 (JRE).
- **High Performance Computing (HPC):**
  - SPEC MPI2007 (MPI), SPEC OMP2012 (OpenMP), SPEC ACCEL (OpenCL, OpenACC).
- **Para PC:**
  - SPECworkstation® 3.1, SYSmark25.

IX. RESUMEN VISUAL (DIAGRAMA CONCEPTUAL)



X. CONSIDERACIONES FINALES

- **No existe un índice único que cumpla todas las propiedades ideales.**
  Cada métrica ofrece una perspectiva parcial y, por ello, la comparación de rendimiento debe hacerse con cargas de prueba bien definidas.

- **La selección de la carga de prueba es crucial:**

    - La carga real es variable y difícil de reproducir, por lo que se recurre a modelos y benchmarks.
    - Los benchmarks permiten comparaciones justas, ya que se siguen reglas estrictas y se estandariza la carga.

- **Ventajas del Benchmarking:**

    - Amplia variedad de herramientas y pruebas para distintos tipos de sistemas y cargas.
    - Permite obtener información valiosa para el diseño y la configuración de nuevos servidores.
