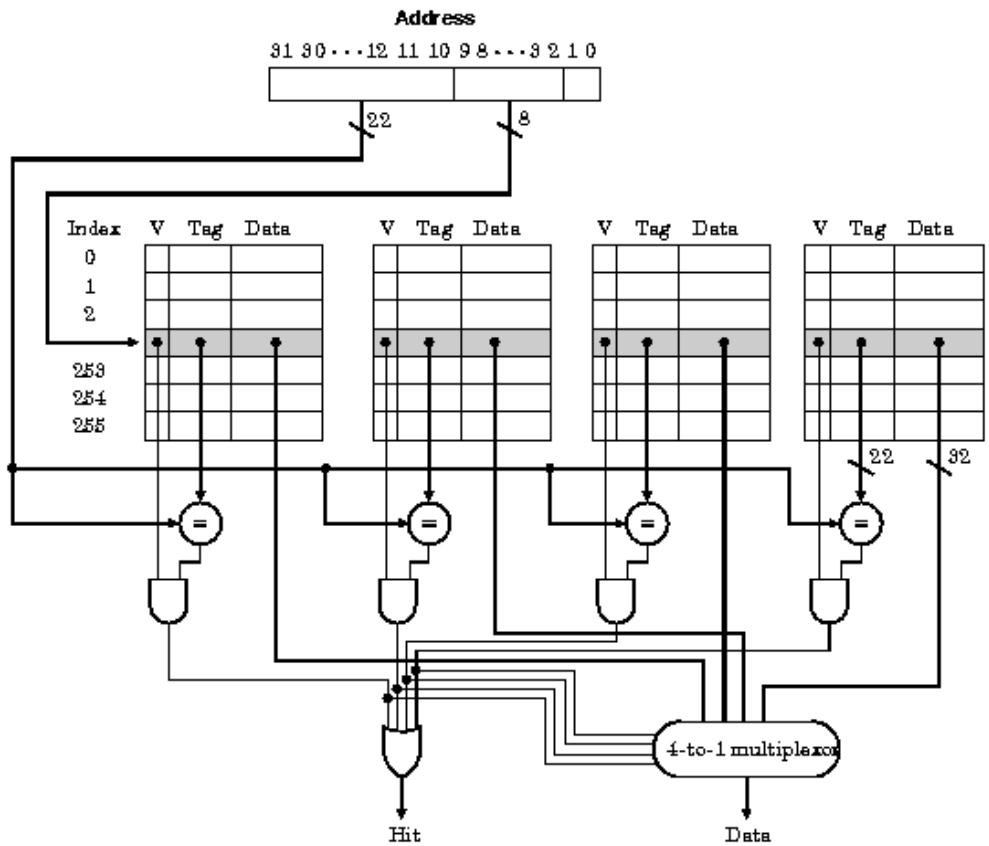
# Presentación de funcionamiento y despliegue

Luis Augusto Sandoval  Gerardo Diaz

# Diagrama resumen del funcionamiento de una cache asociativa por conjuntos de 4 vias



# Resumen de las clases

- **CacheRequest**: Simula las consultas a cache por parte del CPU
- **CacheLine**: Simula las lineas de la cache. Bloques en correspondencia directa o conjuntos en asociativa por conjuntos.
- **PrefetchBuffer**: Simula un buffer de prefetching de adyacencias. Guarda en memoria el bloque siguiente y el anterior a la dirección previa consultada.
- **SetAssociativeCache**: Clase contenedora con la lógica para hacer funcionar las clases anterior y simular el funcionamiento de una memoria cache en conjunto

# Funcionamiento del prefetching

$$bloqueSiguiente = D + (W * B * 8) \quad (1)$$

$$bloqueAnterior = D - (W * B * 8) \quad (2)$$

- **D** es la dirección, **W** el tamaño de la palabra en bytes y **B** el tamaño del bloque en palabras.
- **D** en **bloqueSiguiente** no puede ser mayor a la dirección máxima de la cache ni **D** en **bloqueAnterior** menor a 0
- Si alguna de las 2 condiciones no se cumple, entonces no se tomara el bloque correspondiente

# Benchmark

- **Execution time**: El tiempo de ejecución real del programa. Se dice tiempo real porque descarta el tiempo tomado por el sistema operativo y el kernel.
- **CPU usage**: El tiempo que el sistema Operativo destina a la ejecución del programa, es decir, el tiempo destinado al programa menos el tiempo de CPU que el S.O. destino a la ejecución de otros programas durante la ejecución.
- **RAM usage**: El uso total de RAM por el programa.

# IaC y Terraforming

Para simular el entorno del laboratorio se utilizaron herramientas como Docker y se omitió el uso de librerías externas o no estándares.
La imagen de Docker destinada a la ejecución del programa utiliza como base Ubuntu 14.04, la cual trae consigo Python 3.4.3.
Se evito el uso de librerías no estándares y de módulos o librerías externas.

# Generación de Direcciones

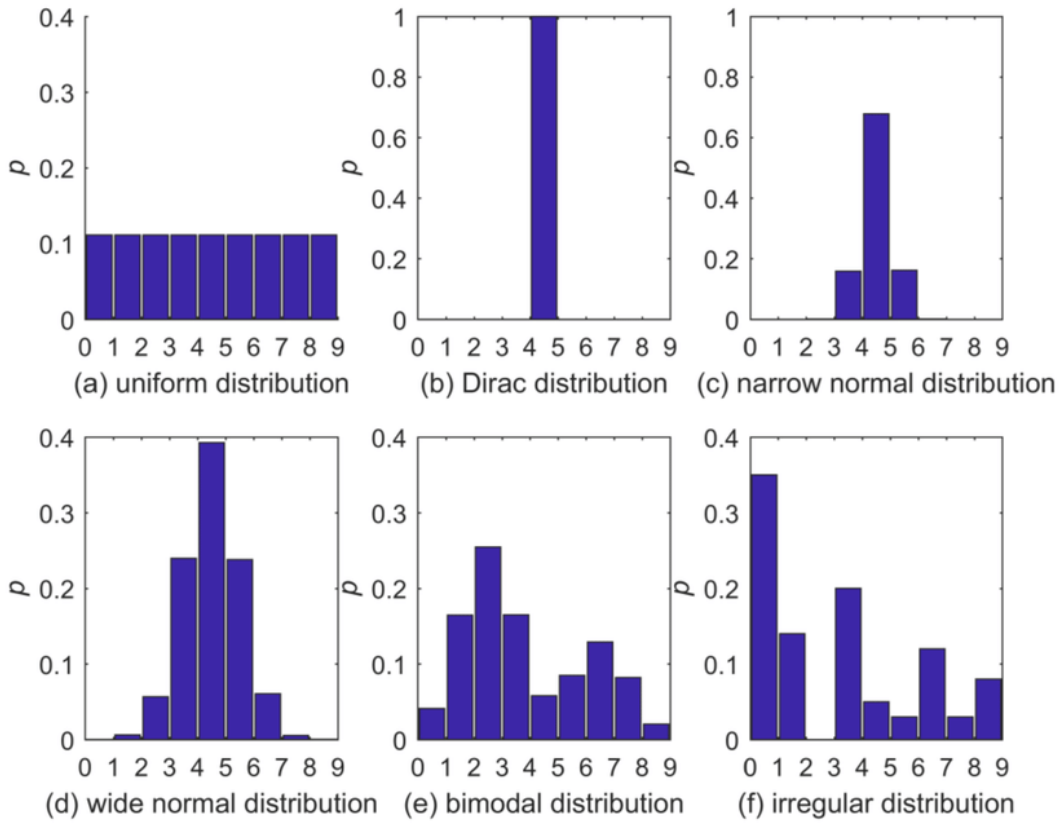



Figura: Tipos de Distribuciones Estándar

# Generación de Direcciones

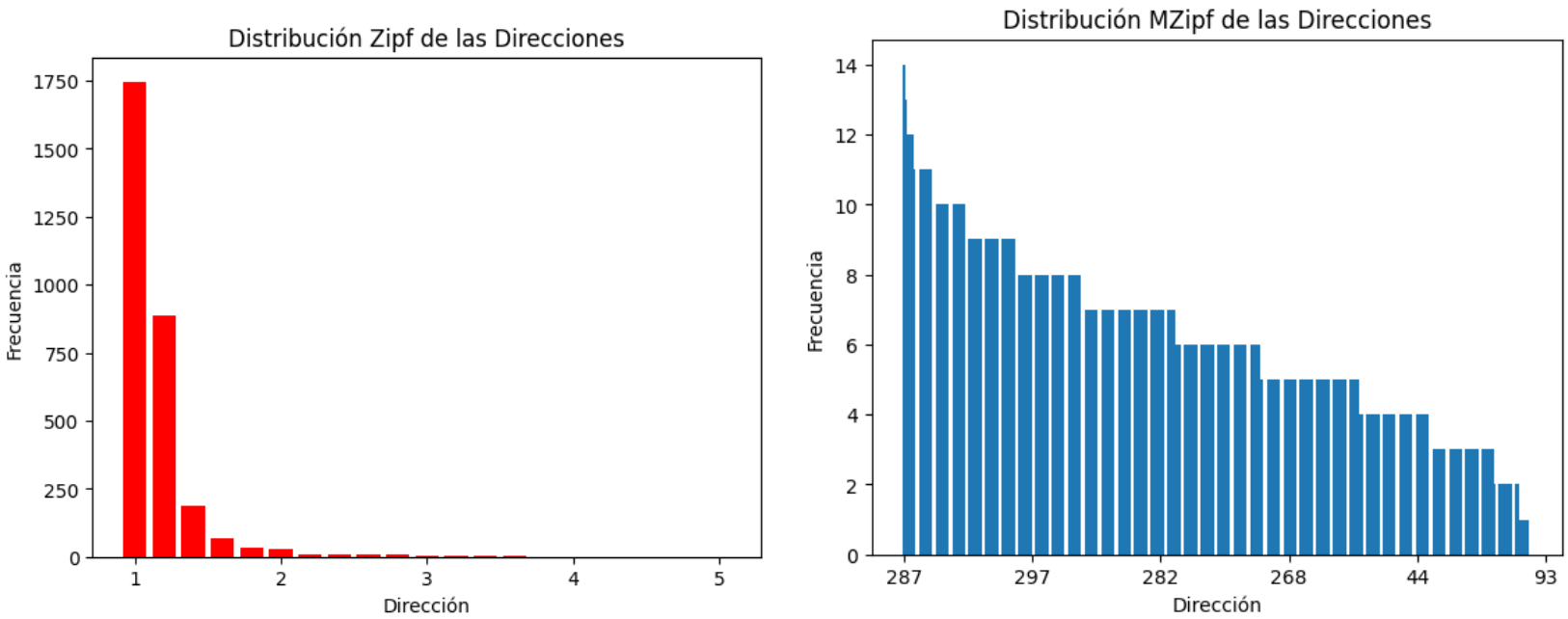
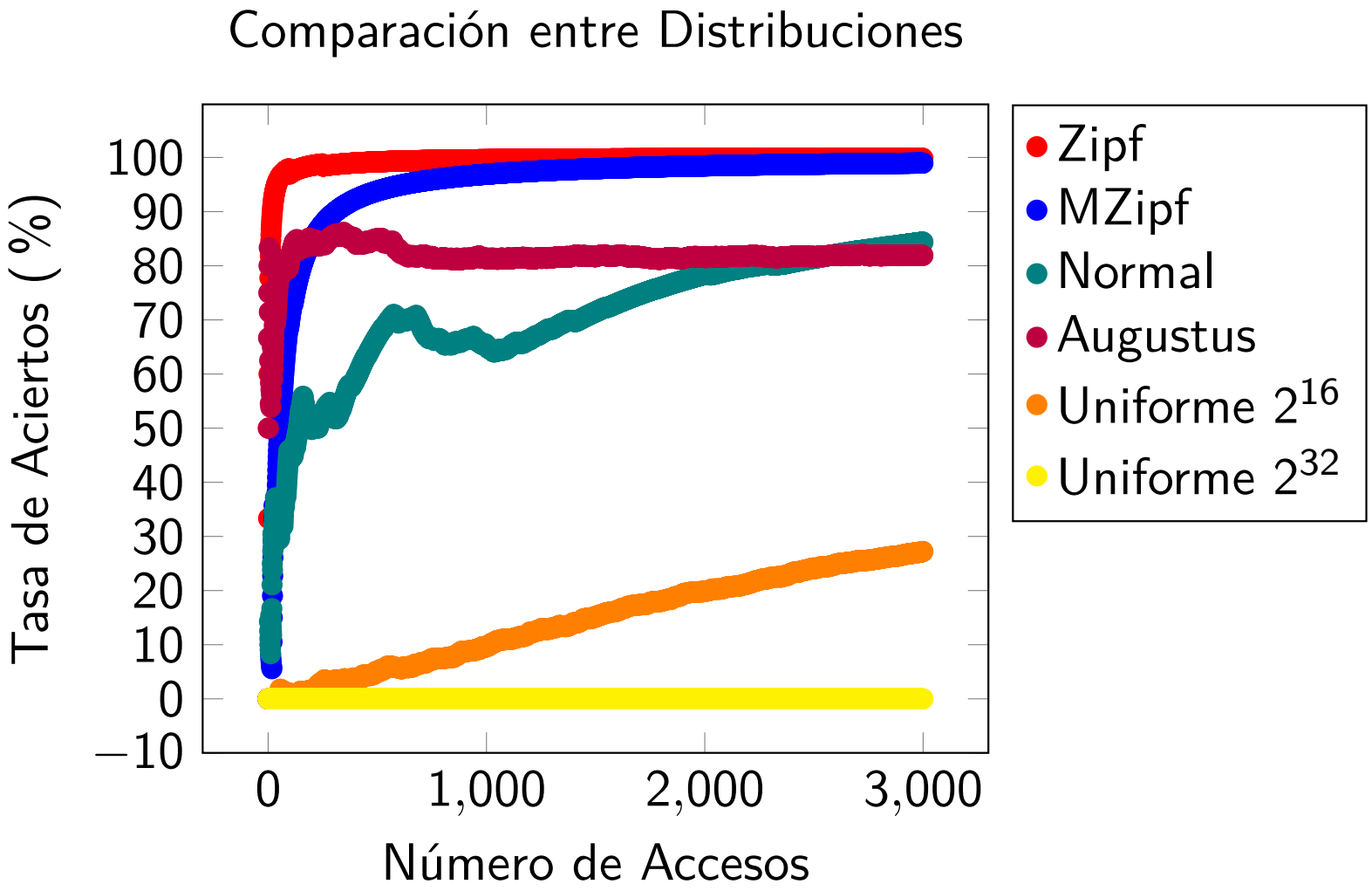



Figura: Distribución Zipf y Modified-Zipf con datos del simulador

# Eficacia del Simulador



# Eficiencia del Simulador

| CPU de 4 núcleos y 3,90 *Ghz* de frecuencia | | | |
|---|---|---|---|
| **Método de Generación** | **Tiempo de Ejecución (*s*)** | **Uso de CPU (%)** | **Uso de RAM (*MB*)** |
| Augustus | 16.70 | 5.51 | 26.87 |
| Uniforme $2^{32}$ | 14.88 | 4.84 | 26.82 |
| Uniforme $2^{16}$ | 14.05 | 4.84 | 25.84 |
| Normal | 17.01 | 5.64 | 25.97 |
| Zipf | 250.92 | 0.28 | 25.89 |
| MZipf | 16.80 | 5.60 | 25.50 |
| CPU de 2 núcleos y 3,20 *Ghz* de frecuencia | | | |
| **Método de Generación** | **Tiempo de Ejecución (*s*)** | **Uso de CPU (%)** | **Uso de RAM (*MB*)** |
| Augustus | 25.66 | 4.21 | 26.57 |
| Uniforme $2^{32}$ | 25.07 | 4.79 | 26.64 |
| Uniforme $2^{16}$ | 25.37 | 4.73 | 25.41 |
| Normal | 25.68 | 4.52 | 25.85 |
| Zipf | 450.06 | 0.40 | 26.19 |
| MZipf | 27.03 | 4.85 | 25.28 |

Tabla: Resultados del *benchmark*